Security Assessment for Orca Whirlpool

Findings and Recommendations Report Presented to:

Orca

January 28, 2022

Version: 0.1

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

Orca engaged Kudelski Security to perform a Security Assessment for Orca Whirlpool.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on January 03 - January 18, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures.
- 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 the risk they pose

- KS-ORCA-WHIRLPOOL-01 – Lamports cannot be transferred from funder accounts
- KS-ORCA-WHIRLPOOL-02 – Code does not compile to BPF
- KS-ORCA-WHIRLPOOL-03 – No error occurs when updating rewards twice at the same time

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

- The team was very supportive and open to discussing the design choices made

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 a Security Assessment for Orca Whirlpool. The following table documents the targets in scope for the engagement. No other systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/orca-so/whirlpool with the commit hash 1bff5eccf12a70002e32627177dde0c502ea2a97.

Files included in the code review

```
orca/
├── programs/
│   └── whirlpool/
│       ├── src/
│       │   ├── constants/
│       │   │   ├── mod.rs
│       │   │   └── test_constants.rs
│       │   ├── instructions/
│       │   │   ├── close_position.rs
│       │   │   ├── collect_fees.rs
│       │   │   ├── collect_protocol_fees.rs
│       │   │   ├── collect_reward.rs
│       │   │   ├── initialize_config.rs
│       │   │   ├── initialize_pool.rs
│       │   │   ├── initialize_reward.rs
│       │   │   ├── initialize_tick_array.rs
│       │   │   ├── liquidity.rs
│       │   │   ├── mod.rs
│       │   │   ├── open_position.rs
│       │   │   ├── set_collect_protocol_fees_authority.rs
│       │   │   ├── set_default_fee_rate.rs
│       │   │   ├── set_default_protocol_fee_rate.rs
│       │   │   ├── set_fee_authority.rs
│       │   │   ├── set_fee_rate.rs
│       │   │   ├── set_protocol_fee_rate.rs
│       │   │   ├── set_reward_authority.rs
│       │   │   ├── set_reward_authority_by_super_authority.rs
│       │   │   ├── set_reward_emissions.rs
│       │   │   ├── set_reward_emissions_super_authority.rs
│       │   │   ├── swap.rs
│       │   │   └── update_fees_and_rewards.rs
│       │   ├── manager/
│       │   │   ├── mod.rs
│       │   │   ├── position_manager.rs
│       │   │   ├── tick_manager.rs
│       │    │    └── whirlpool_manager.rs
│       │    ├── math/
│       │    │    ├── bit_math.rs
│       │    │    ├── bn.rs
│       │    │    ├── liquidity_math.rs
│       │    │    ├── mod.rs
│       │    │    ├── swap_math.rs
│       │    │    ├── tick_math.rs
│       │    │    └── token_math.rs
│       │    ├── state/
│       │    │    ├── config.rs
│       │    │    ├── mod.rs
│       │    │    ├── position.rs
│       │    │    ├── tick.rs
│       │    │    └── whirlpool.rs
│       │    ├── util/
│       │    │    ├── mod.rs
│       │    │    ├── swap_ticks.rs
│       │    │    ├── token.rs
│       │    │    └── util.rs
│       │    ├── errors.rs
│       │    └── lib.rs
│       ├── Cargo.toml
│       └── Xargo.toml
├── Anchor.toml
├── Cargo.lock
└── Cargo.toml
```

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Security Assessment for Orca Whirlpool, we discovered:

- 3 findings with a LOW severity rating.

The following chart displays the findings by severity.

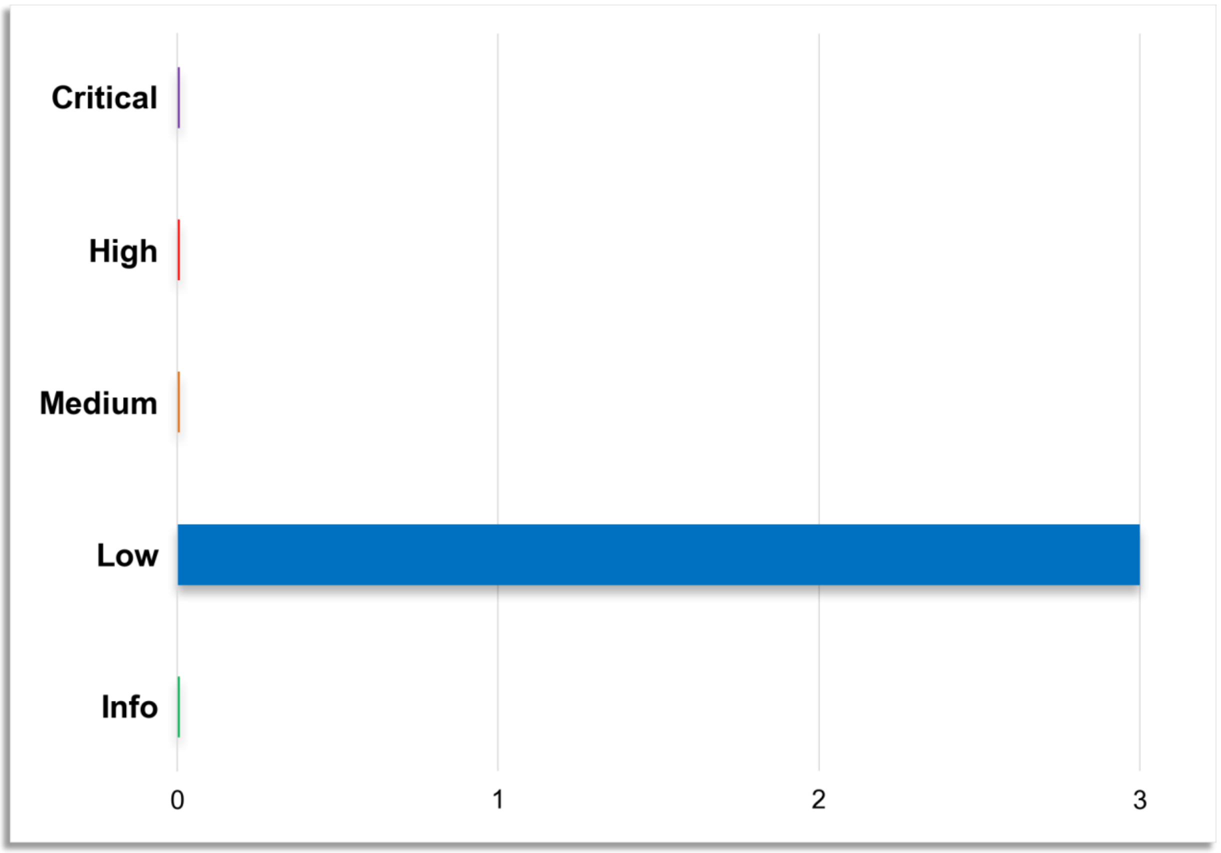

Figure 1: Findings by Severity.

Findings

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

The following table provides an overview of the findings.

#	Severity	Description
KS-ORCA-WHIRLPOOL-01	Low	Lamports cannot be transferred from funder accounts
KS-ORCA-WHIRLPOOL-02	Low	Code does not compile to BPF
KS-ORCA-WHIRLPOOL-03	Low	No error occurs when updating rewards twice at the same time

Table 2: Findings Overview

Technical analysis

Based on the source code, the following account relationship graphs or reference graphs were made to verify the validity of the code and confirm that the intended functionality was implemented correctly and 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:

- Potential panics were detected but posed no security risk
- Potential errors regarding wraps/unwrap, expect, and wildcards were detected but pose no security risk
- No internal unintentional unsafe references

Authorization

The review used relationship graphs to show the relations between account input passed to the program's instructions. 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 an account of an attacker's choosing and thus pose a security risk.

In particular, the graphs will show if signing accounts are referred to. If a signing account is not referred to, any account can be used to sign the transaction, causing insufficient authorization.

No insufficient authorization was found based on the analyzes of the relationship graphs. For details, see section Relationship Graphs starting on page 18.

Conclusion

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

Technical Findings

General Observations

Orca Whirlpool is a Solana smart contract implementation of an Automated Market Maker (AMM).

The code is beautifully structured. The struct and handler functions are bundled in a single file for each instruction, making it easy to get an overview. This has a very positive impact on the maintainability of the code.

The code is implemented using Serum's Anchor framework. Anchor automates many validation checks that otherwise are easy to forget during development. As can be seen in the relationship graphs, account validation is sufficiently implemented for the instructions of the Whirlpool program.

Unit tests have been implemented to cover 61% of the lines. This is an excellent start. Tests' code not sufficiently covered include the `lib.rs` and code in the `instructions` folder. It is recommended to implement functional tests using the `anchor-client` and the `Solana-program-test` crate to cover the code for instruction handling. Inspiration for writing functional tests can be found in the repository for `Solana-program-test`.

It is good practice to return errors on invalid input. This makes the code more maintainable as present assumptions may not hold in the future if the code is being updated. Error handling could be improved as panic may occur on various occasions:

- `unwrap` is used 25 times to do unchecked unwrapping of a `Result`. In some cases, functions are implemented to return `Result` objects even though there are no cases where errors occur.
- Unchecked integer division is used 7 times.
- Unchecked integer modulo is used 3 times.

The reviewed code contained 46 TODOs. As some of these TODOs concern input validation to avoid integer overflows, it is highly recommended to handle all TODOs before going into production.

The implementation uses dynamic allocation of the allocated ticks. This is similar to implementing order books in Serum DEX, even though it is more complex. The math implementation looks valid, although it contains several TODOs preventing integer overflows.

It should be considered if the use of SPL Tokens for position authorities is worth the price. A unique mint is created for that specific position when a position is opened. Only one token is minted. The position mint token is then used to prove the authority of the position. However, this is a single reference. But it is modeled using four accounts. The position account refers to the mint account. The token account refers to the mint account and the token owner, the position authority. A shortcut could be to have the position account directly refer to the position authority account. This requires less rent, and all accounts can be closed, and invested rent can be repaid. The trade-off is that this will allow closed positions to be reinitialized, and it must be ensured that reinitialization cannot be used as an attack vector.

The small number of findings indicates a high maturity of the code. The overall impression is that the code is well-written, secure, and maintainable.

Lamports cannot be transferred from funder accounts

Finding ID: KS-ORCA-WHIRLPOOL-01
Severity: **Low**
Status: **Open**

Description

The signing `funder` account used for the `initialize_config`, `initialize_pool`, `initialize_tick_array`, `initialize_reward`, and `open_position` instructions is not specified as writable.

The funder account is used to "fund" allocation of the accounts during the initialization instructions by invoking system instructions—the system instructions transfer lamports from the funder accounts. Transfer of lamports changes the account state and requires the account to be writable to succeed.

> The balance of read-only and executable accounts may not change.

- Solana Documentation - Runtime - Policy

Proof of Issue

Identical code is used to specify the `funder` account for the different instructions:

- **Filename:** instructions/initialize_config.rs
  Line number: 9
- **Filename:** instructions/initialize_pool.rs
  Line number: 13
- **Filename:** instructions/initialize_tick_array.rs
  Line number: 9
- **Filename:** instructions/initialize_reward.rs
  Line number: 12
- **Filename:** instructions/open_position.rs
  Line number: 9

```
    pub funder: Signer<'info>,
```

Severity and Impact Summary

Anchor accounts are writable if they are annotated with constraints for initializing with `init` or mutable with `mut`. If not specified, an account will be read-only. This means that any changes to the input data will not be written to the account on the blockchain.

Furthermore, the IDL specification generated by Anchor will not specify the define the `funder` accounts as writable. Consequently, clients automatically generated using either the Node.js library `@project-serum/anchor` or the Rust crate `anchor-client` will not pass the `funder` account as writable, making client transactions fail.

A small gap in the Solana blockchain will allow these instructions to succeed. If the `funder` account is also used to pay the transaction fee, it will be marked as writable.

As such, this does not pose a security issue. However, it is an annoyance that auto-generated clients cannot call the initialization instructions unless the same account is used to fund the initializing accounts and pay the transaction fees.

Recommendation

Annotate all the `funder` accounts with the Anchor `mut` constraint to make it writable:

```
    #[account(mut)]
    pub funder: Signer<'info>,
```

References

- Solana Documentation - Runtime - Policy
- docs.rs - anchor-lang::Accounts - Normal Constraints
- docs.rs - anchor-client
- npm - @project-serum/anchor

Code does not compile to BPF

Finding ID: KS-ORCA-WHIRLPOOL-02
Severity: **Low**
Status: **Open**

Description

The reviewed code cannot be compiled with the Solana BPF as the target.

Proof of Issue

The code has been built using Rust version 1.58.0 and Solana toolchain version 1.8.1.

Executing `anchor build` gives the following output:

```
BPF SDK: /home/audit/.local/share/solana/install/releases/1.8.1/solana-
release/bin/sdk/bpf
Running: rustup toolchain list -v
Running: cargo +bpf build --target bpfel-unknown-unknown --release
   Compiling whirlpool v0.1.0 (/home/audit/orca/programs/whirlpool)
warning: unused import: `super::U256`
  --> programs/whirlpool/src/math/bn.rs:94:9
   |
94 |     use super::U256;
   |         ^^^^^^^^^^^
   |
   = note: `#[warn(unused_imports)]` on by default

warning: unused import: `crate::errors::ErrorCode`
  --> programs/whirlpool/src/math/bn.rs:95:9
   |
95 |     use crate::errors::ErrorCode;
   |         ^^^^^^^^^^^^^^^^^^^^^^^^

error[E0658]: use of unstable library feature 'array_map'
  --> programs/whirlpool/src/manager/tick_manager.rs:59:34
   |
59 |                         reward_infos.map(|reward_info|
reward_info.growth_global_x64),
   |                                  ^^^
   |
   = note: see issue #75243 <https://github.com/rust-lang/rust/issues/75243>
for more information
   = help: add `#![feature(array_map)]` to the crate attributes to enable

error: aborting due to previous error; 2 warnings emitted

For more information about this error, try `rustc --explain E0658`.
error: could not compile `whirlpool`

To learn more, run the command again with --verbose.
```

Severity and Impact Summary

Not compiling the code with the Solana BPF as the target will prevent deployment to the Solana blockchain.

Recommendation

The `reward_infos` array, which is mapped, is of fixed size. The simplest solution is to write the mapping by hand.

So, change the code from:

Filename: manager/tick_manager.rs
Line number: 59

```
                        reward_infos.map(|reward_info|
reward_info.growth_global_x64),
```

into

```
                        [
                            reward_infos[0].growth_global_x64,
                            reward_infos[1].growth_global_x64,
                            reward_infos[2].growth_global_x64,
                        ],
```

No error occurs when updating rewards twice at the same time

Finding ID: KS-ORCA-WHIRLPOOL-03
Severity: **Low**
Status: **Open**

Description

The `next_whirlpool_reward_infos` function calculates the following global reward growth variables based on the given timestamp. It verifies that the `next_timestamp` is not before the `reward_last_updated_timestamp`. But it allows the timestamps to be the same.

Proof of Issue

Filename: manager/whirlpool_manager.rs
Line number: 12

```
    let curr_timestamp = whirlpool.reward_last_updated_timestamp;
    if next_timestamp < curr_timestamp {
        return Err(ErrorCode::InvalidTimestamp.into());
    }
```

The `next_timestamp` is passed from the clock sysvar several times in the code:

```
    let timestamp = to_timestamp_u64(clock.unix_timestamp)?;
```

If a transaction repeats the multiple instructions that trigger the `next_whirlpool_reward_infos` function, the timestamp will be the same as it is bound to the atomic processing of the transaction.

Luckily, the rewards are calculated using the difference between the next and the current timestamps:

Filename: manager/whirlpool_manager.rs
Line number: 24, 34

```
    let time_delta = u128::from(next_timestamp - curr_timestamp);
    for i in 0..NUM_REWARDS {
# ...
        let reward_growth_delta = mul_div(
            time_delta,
            reward_info.emissions_per_second_x64,
            whirlpool.liquidity.try_into().unwrap(),
        )
        .unwrap_or(0);
```

So, when the timestamps are the same, `time_delta` will be `0`, and the reward will be `0`:

Severity and Impact Summary

This issue has no impact as no rewards will be returned when no time has passed since the last reward update.

However, the behavior is unexpected as the `next_whirlpool_reward_infos` function returns an error when the next timestamp is before the timestamp of the last reward update.

The same behavior would be expected when the timestamp is not after the last reward update.

Recommendation

Implemented the same behavior for the `next_whirlpool_reward_infos` function when the timestamp is not after the last reward update.

When the next timestamp is less than or equal to the timestamp of the last reward update either:

1. Return an error.
2. Or return no rewards.

Relationship Graphs

A relationship graph shows relational requirements to the input of an instruction. It does not show how the instruction modifies accounts.

Relationship graphs are used to analyze insufficient authorization and unchecked account relations. Insufficient authorization allows unintended access. Unchecked account relations may allow account and data injection resulting in unintended behavior and access.

Color styles have been added relationship graphs to highlight unique relationships:

- Rounded boxes indicate account input. Other boxes are data input or constants.
- Arrows indicate required relations between accounts.
- Arrows on a dashed line indicate implicitly required relations. For example, relations are required by a CPI. This is emphasized as the program cannot guarantee the behavior of external programs.
- Thick blue borders indicate required signers.
- Green boxes indicate accounts required to be owned by the whirlpool program.
- Purple boxes indicate accounts required to be owned by the SPL token program.
- Cyan boxes indicate accounts required to be owned by the system program.
- PDA calculations are indicated as hexagons. The PDA seeds are held together by a square with double-lined borders.
- Red boxes indicate unchecked account or data input. Data input is always unchecked as it does not originate from the blockchain and may open for data injection attacks. Usage of unchecked input should always be verified.

Initialize Config

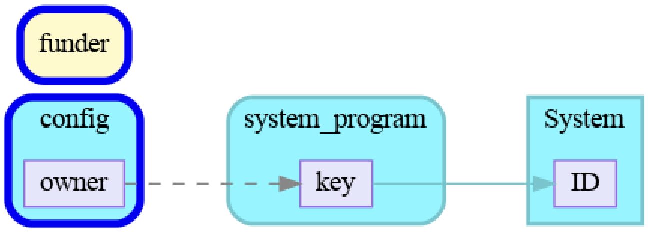

Figure 2: Relationship graph for InitializeConfig instruction.

Initialize Pool

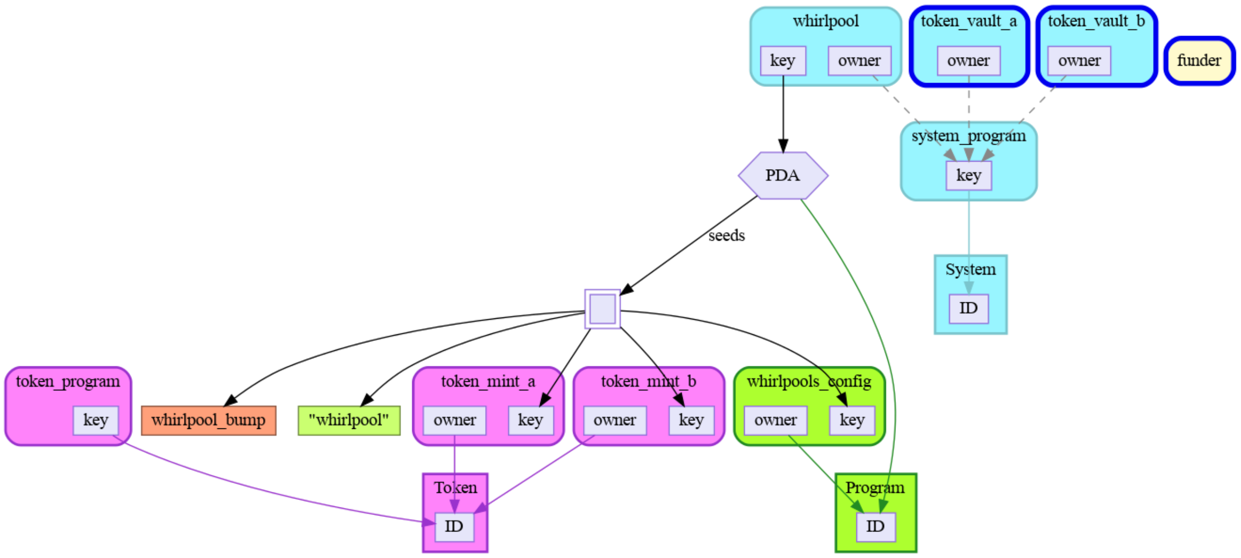

Figure 3: Relationship graph for InitializePool instruction.

The `whirlpool_bump` is passed as data input and used as part of the seed for generating the PDA for the `whirlpool` account. As Anchor validates that the `whirlpool_bump` is equal to the bump seed output from the `find_program_address` function, it is impossible to abuse this.

Initialize Tick Array

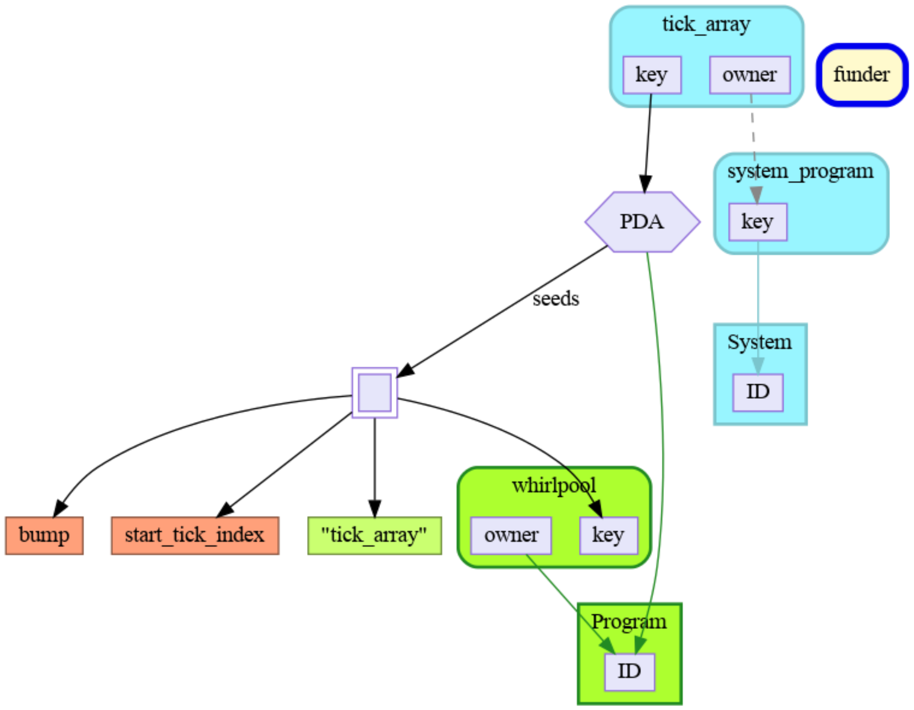

Figure 4: Relationship graph for InitializeTickArray instruction.

The `bump` is passed as data input and used as the seed for generating the PDA for the `tick_array` account. As Anchor validates that the `bump` is equal to the bump seed output from the `find_program_address` function, it is impossible to abuse this.

The `start_tick_index` is passed as data input. Bypassing different values of `start_tick_index`es it is possible to create multiple tick arrays for a single `whirlpool` account. According to Orca, this is intentional.

Initialize Reward

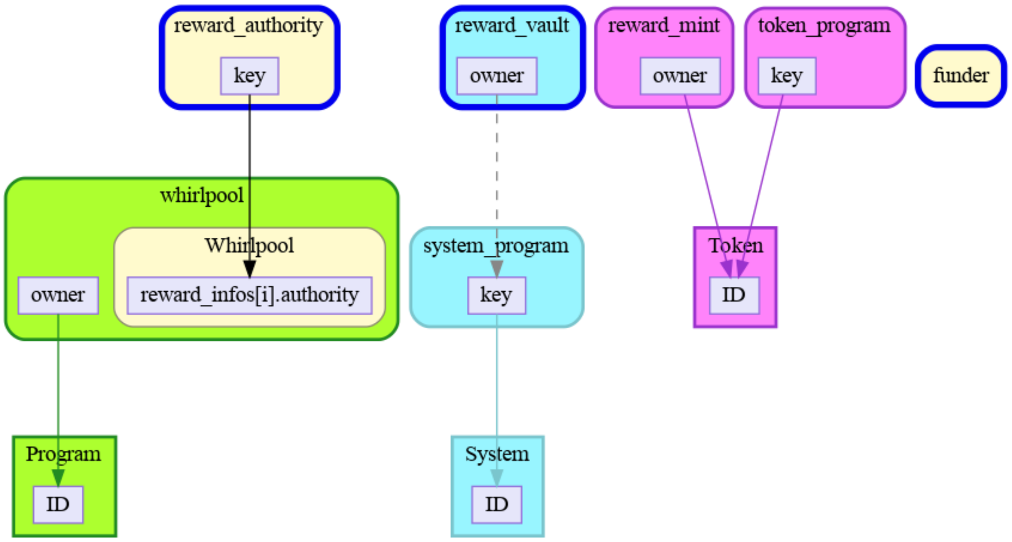

Figure 5: Relationship graph for InitializeReward instruction.

Open Position

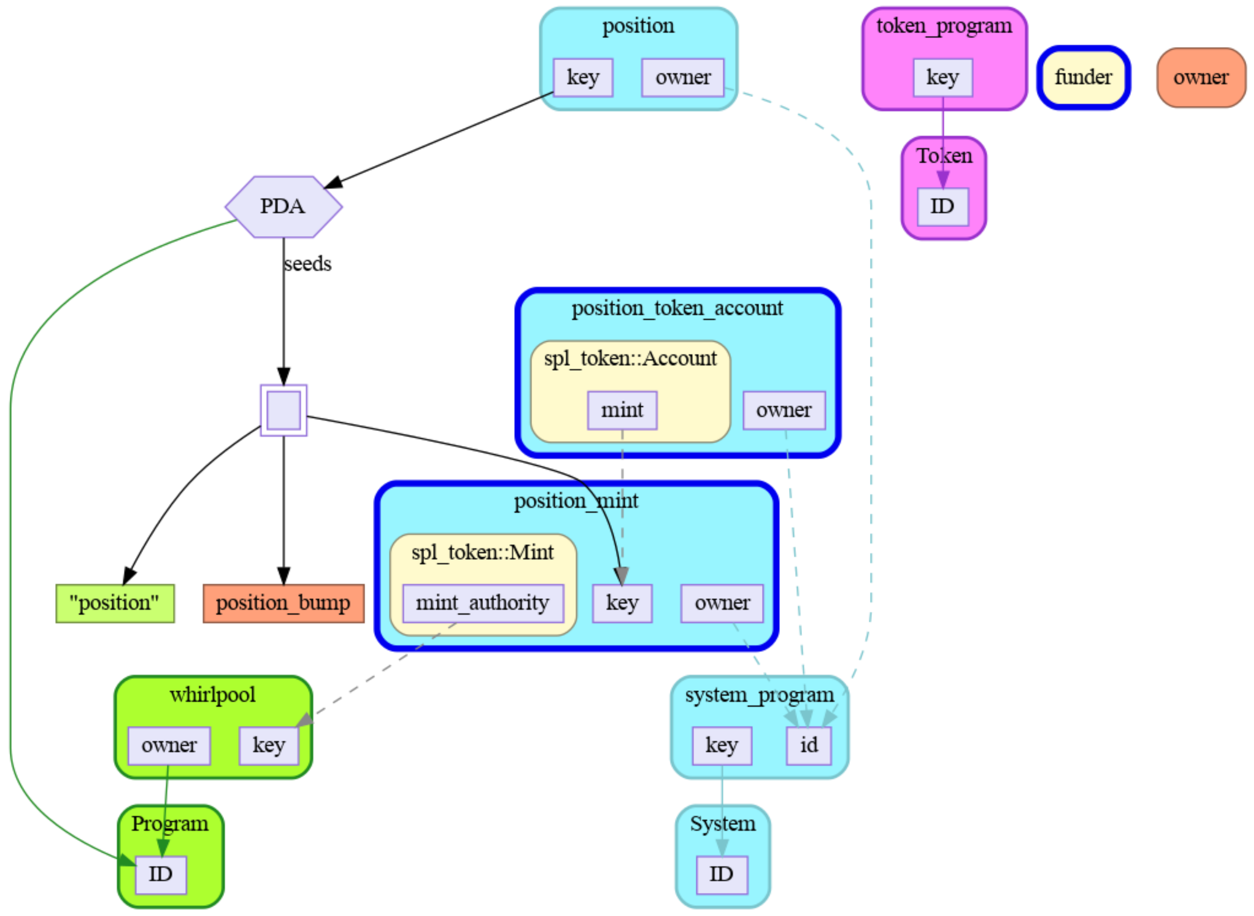

Figure 6: Relationship graph for OpenPosition instruction.

The `position_bump` is passed as data input and used as the seed for generating the PDA for the `position` account. Again, Anchor validates that the `position_bump` equals the bump seed output from the `find_program_address` function. It is not possible to abuse this.

The `owner` is passed as unchecked account input and referenced as the owner of the initialized `position_token_account`. In other instructions, as proof of authorization, the `owner` account must sign, and the balance of the `position_token_account` must be the only single `position_mint` token created.

Close Position

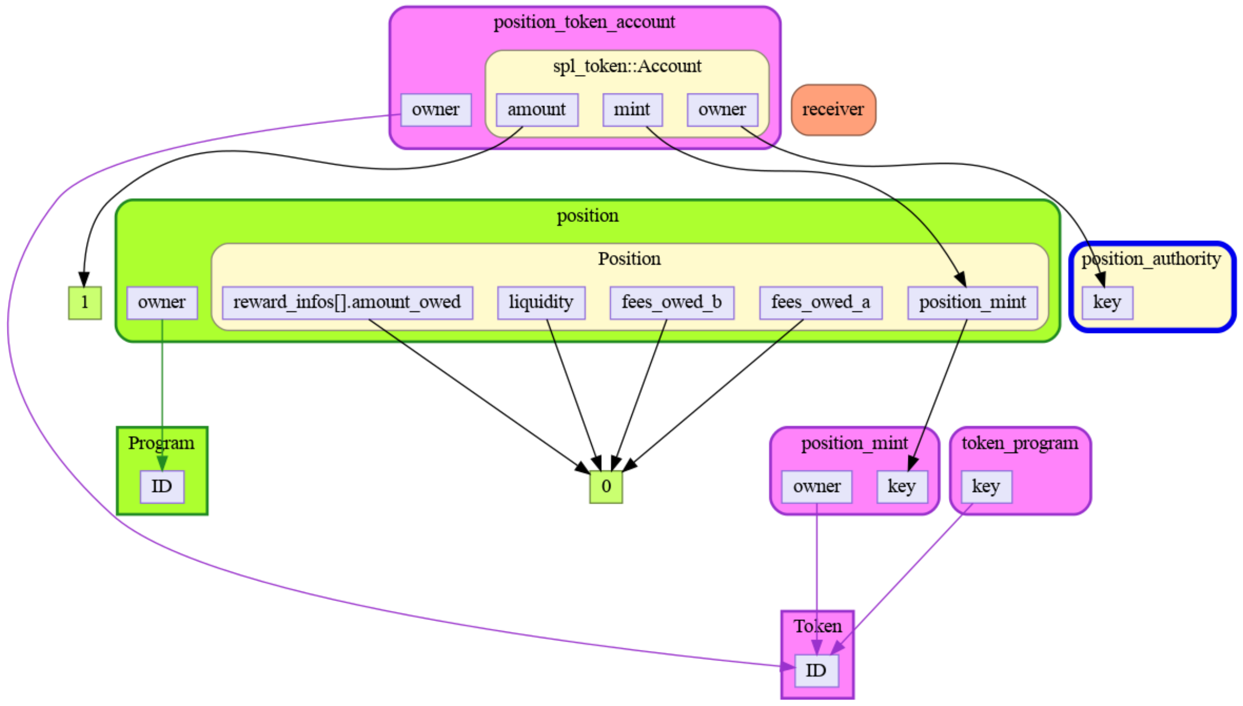

Figure 7: Relationship graph for ClosePosition instruction.

The `receiver` is passed as unchecked account input and used to transfer the remaining rent from the `position` account being closed. This is acceptable as authorization requires the `position_authority` to sign and be the owner of the `position_token_account,` which must hold the only single `position_mint` token created.

Increase Liquidity

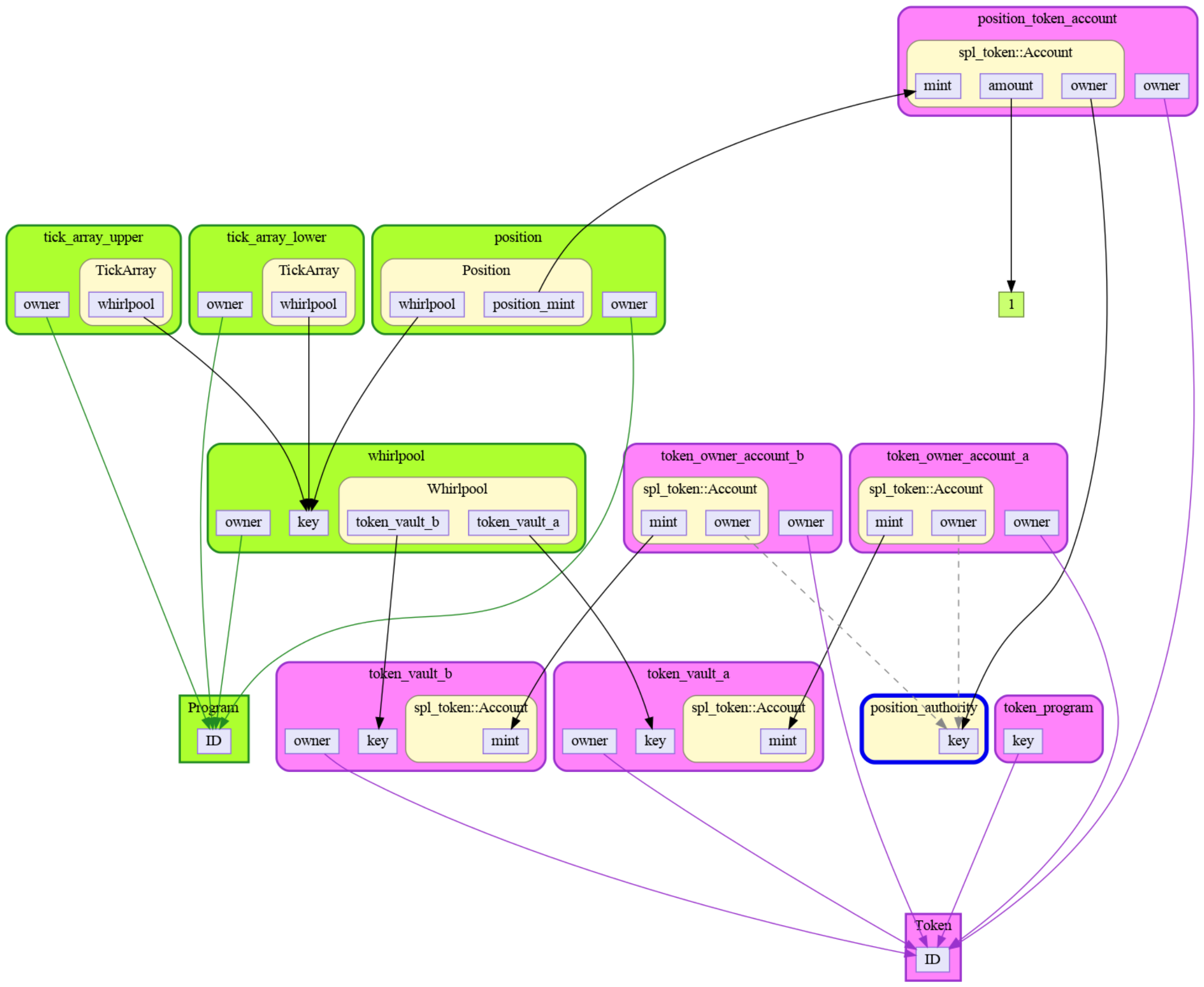

Figure 8: Relationship graph for IncreaseLiquidity instruction.

Decrease Liquidity

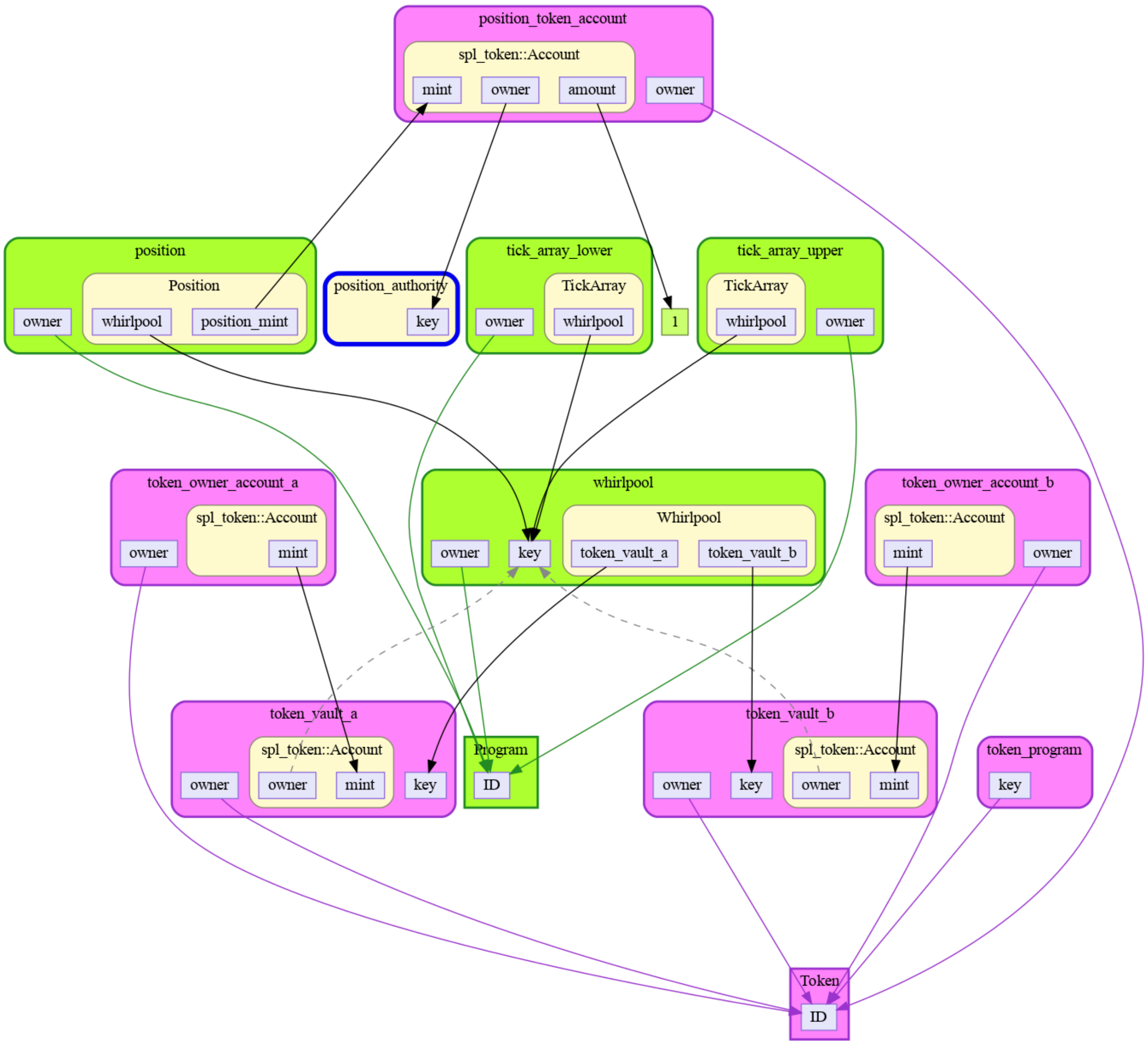

Figure 9: Relationship graph for DecreaseLiquidity instruction.

Set Reward Emissions

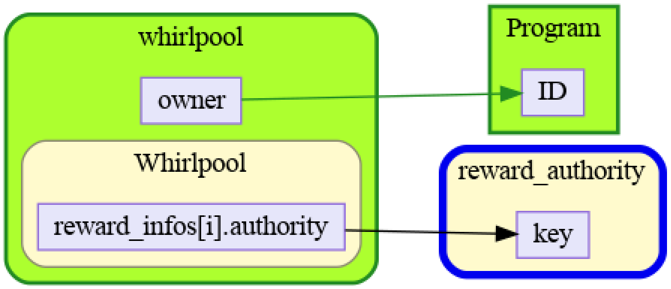

Figure 10: Relationship graph for SetRewardEmissions instruction.

Update Fees and Rewards

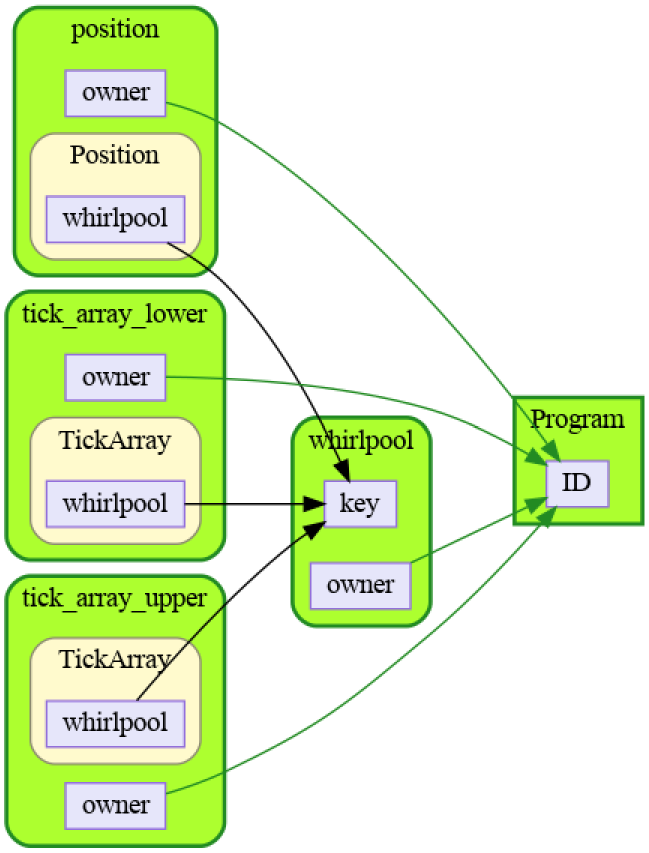

Figure 11: Relationship graph for UpdateFeesAndRewards instruction.

Collect Fees

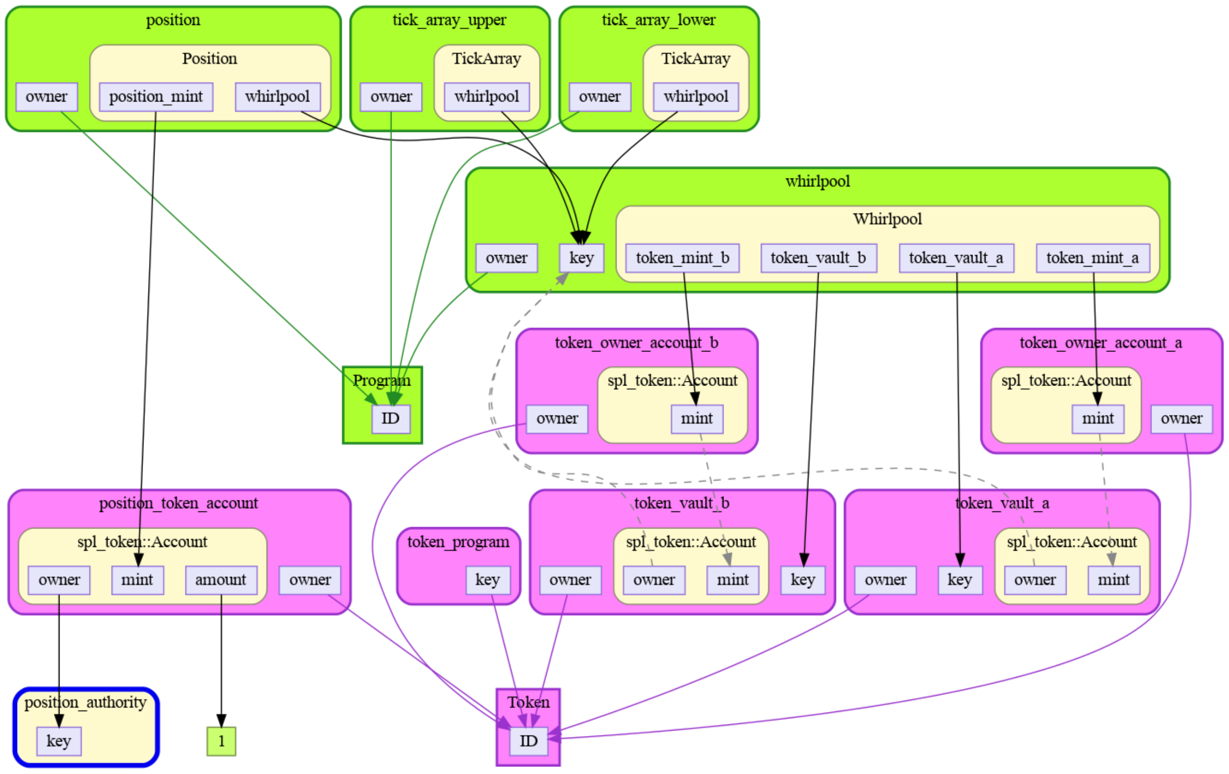

Figure 12: Relationship graph for CollectFees instruction.

Collect Reward

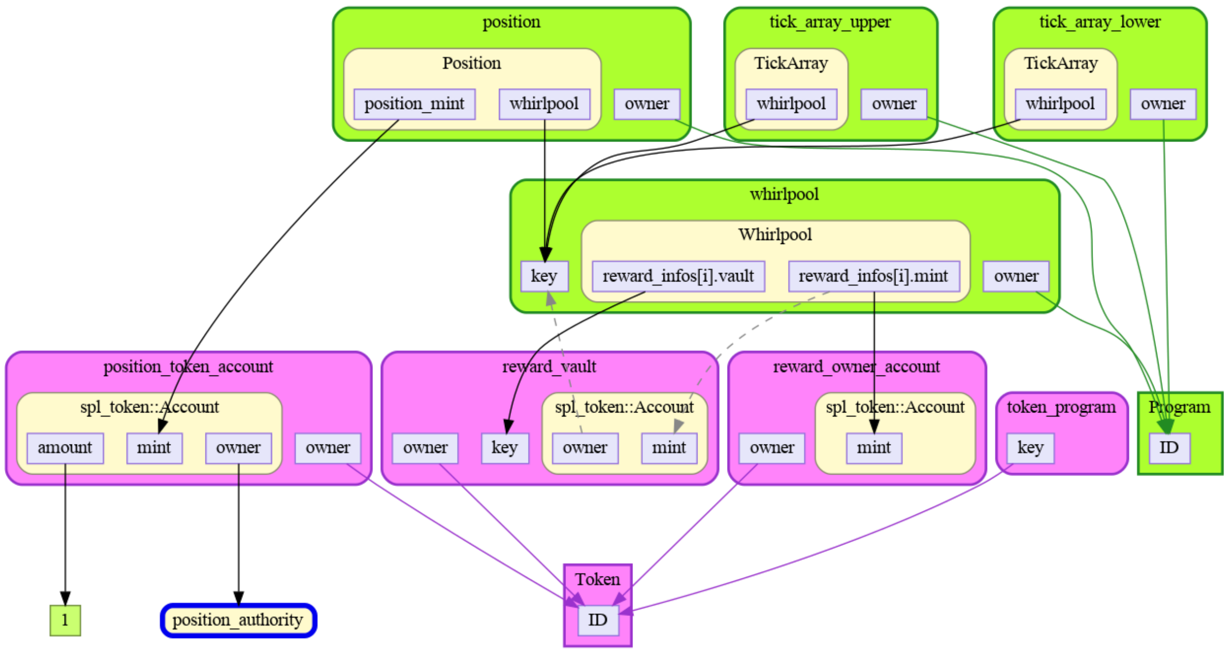

Figure 13: Relationship graph for CollectReward instruction.

Collect Protocol Fees

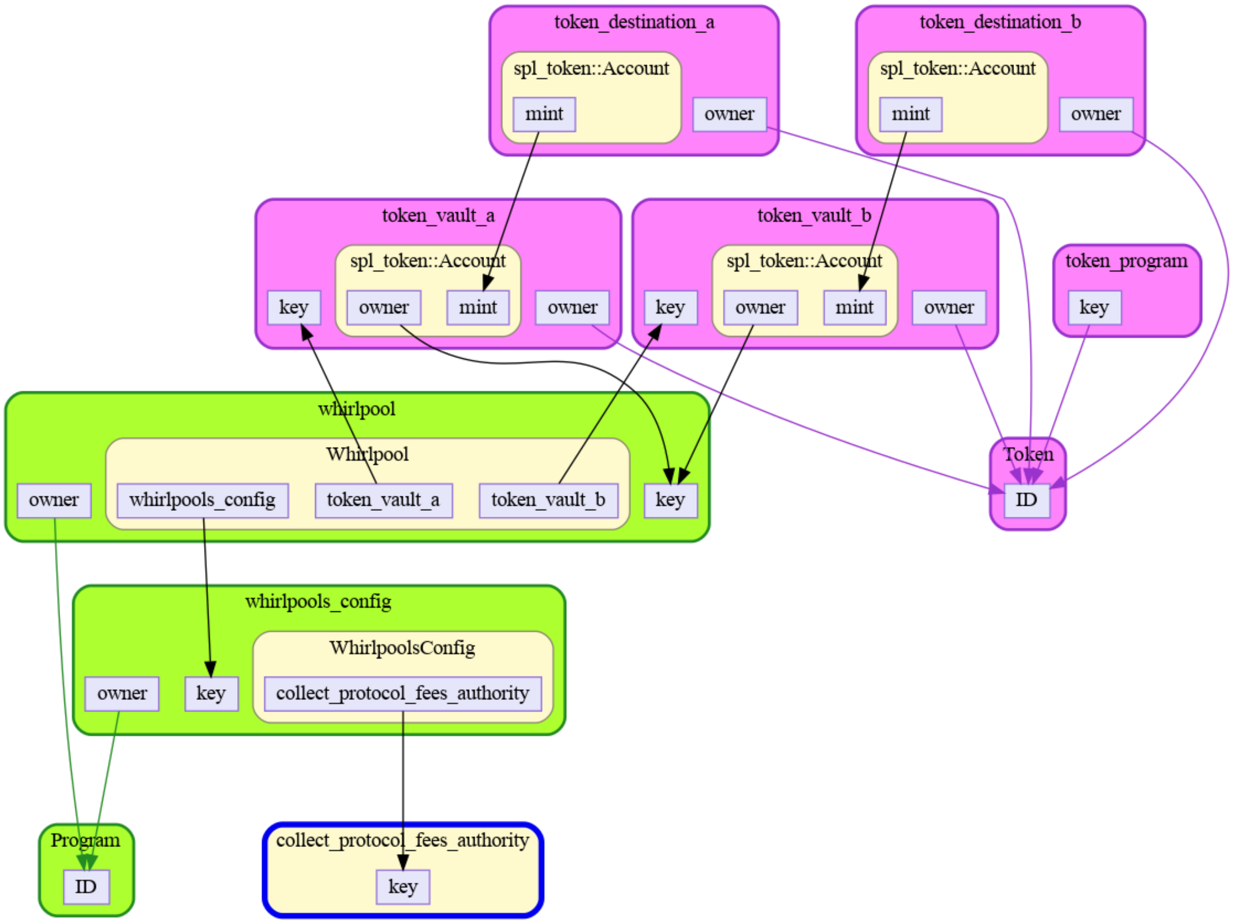

Figure 14: Relationship graph for CollectProtocolFees instruction.

Swap

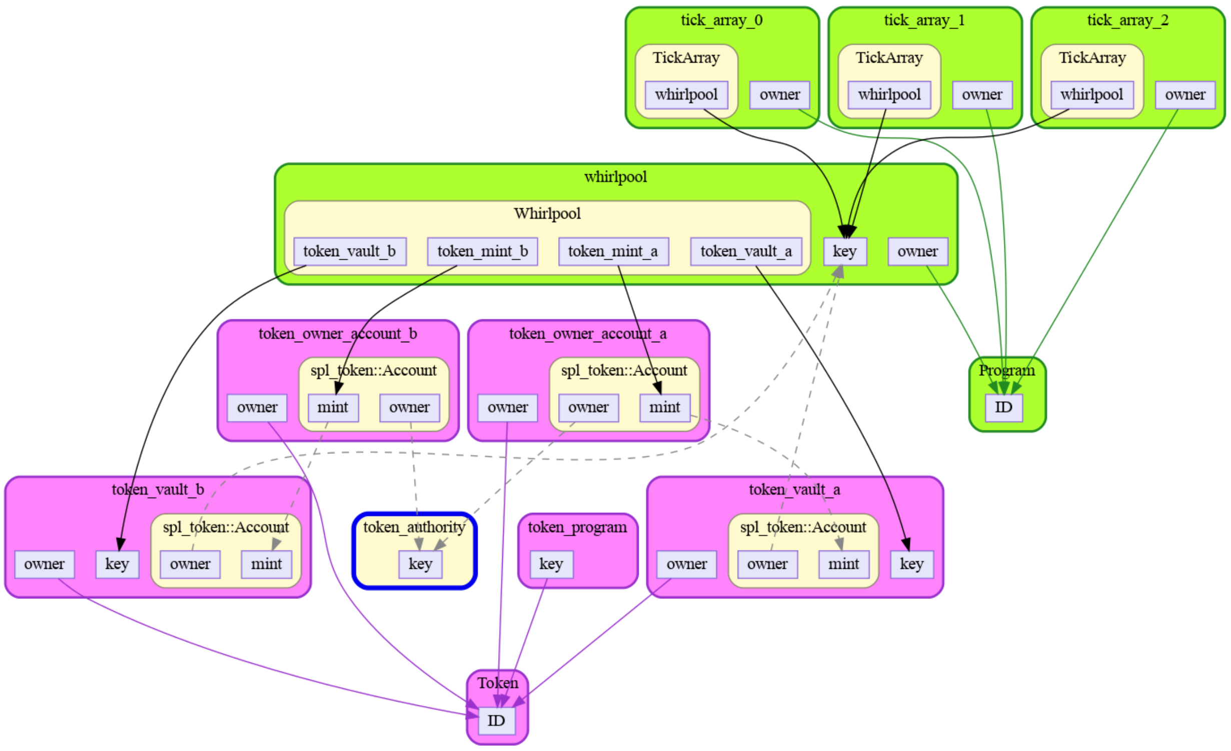

Figure 15: Relationship graph for Swap instruction.

Set Default Fee Rate

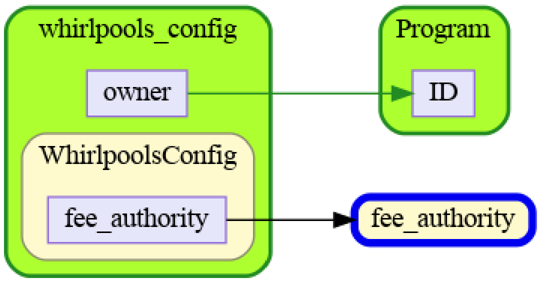

Figure 16: Relationship graph for SetDefaultFeeRate instruction.

Set Default Protocol Fee Rate

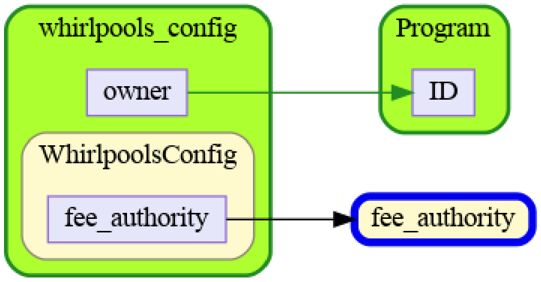

Figure 17: Relationship graph for SetDefaultProtocolFeeRate instruction.

Set Fee Rate

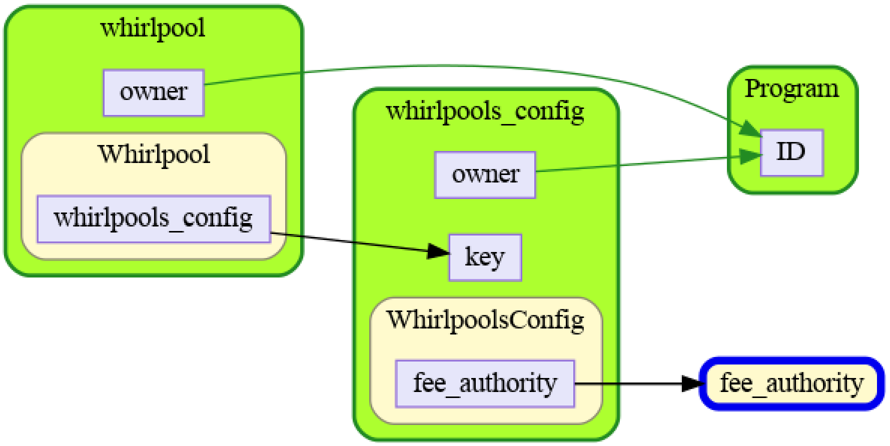

Figure 18: Relationship graph for SetFeeRate instruction.

Set Protocol Fee Rate

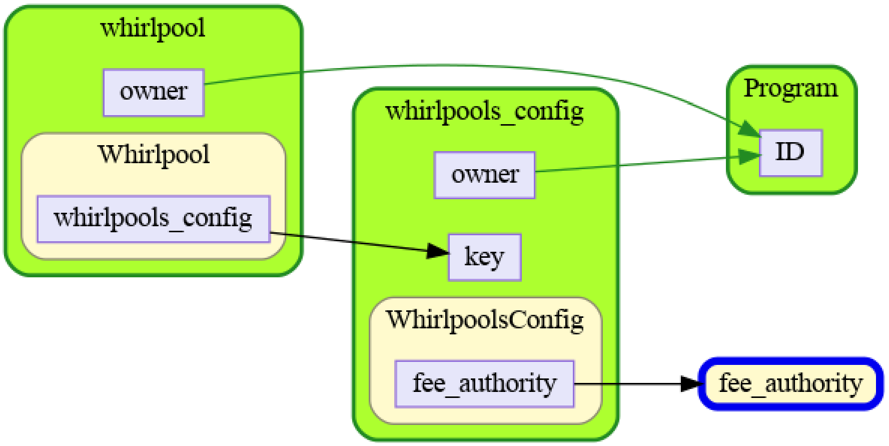

Figure 19: Relationship graph for SetProtocolFeeRate instruction.

Set Fee Authority

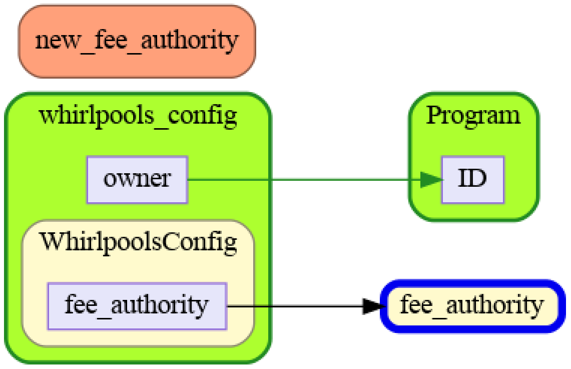

Figure 20: Relationship graph for SetFeeAuthority instruction.

The `new_fee_authority` is passed as unchecked account input to replace the `fee_authority` of the `whirlpools_config`. The existing `fee_authority` is required to sign for a suitable replacement.

Set Collect Protocol Fees Authority

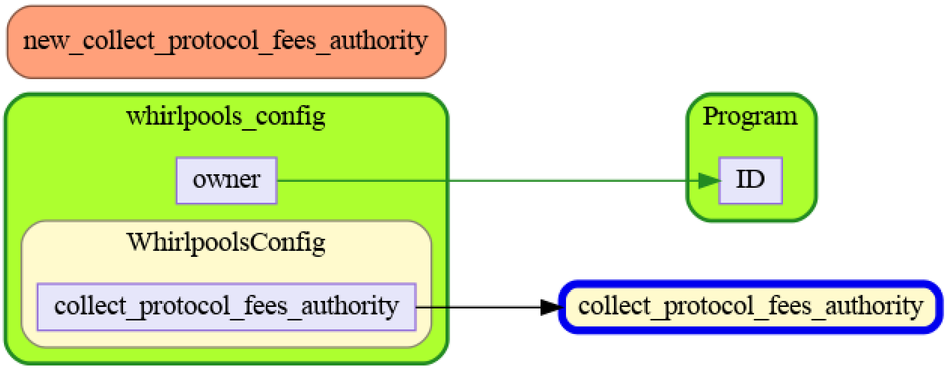
new_collect_protocol_fees_authority
whirlpools_config
owner
WhirlpoolsConfig
collect_protocol_fees_authority
Program
ID
collect_protocol_fees_authority

Figure 21: Relationship graph for SetCollectProtocolFeesAuthority instruction.

The `new_collect_protocol_fees_authority` is passed as unchecked account input to replace the `collect_protocol_fees_authority` of the `whirlpools_config`. The existing `collect_protocol_fees_authority` is required to sign for a suitable replacement.

Set Reward Authority

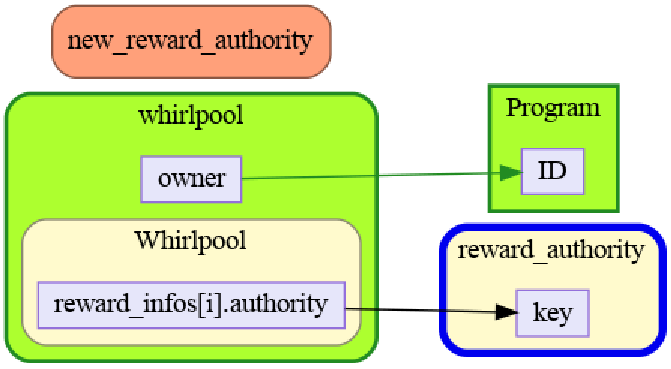

Figure 22: Relationship graph for SetRewardAuthority instruction.

The `new_reward _authority` is passed as unchecked account input to replace the `authority` of one of `whirlpool`'s `reward_info` references. The existing `authority` is required to sign for a suitable replacement.

Set Reward Authority by Super Authority

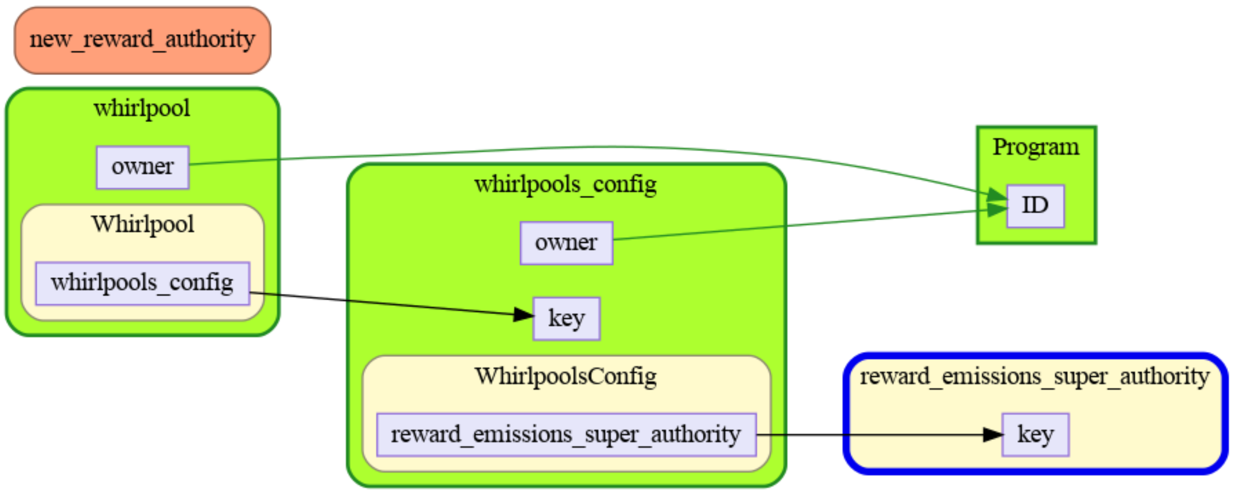

Figure 23: Relationship graph for SetRewardAuthorityBySuperAuthority instruction.

The `new_reward_authority` is passed as unchecked account input to replace the `reward_authority` of one of `whirlpool`'s `reward_info` references. The existing `whirlpools_config`'s `reward_emissions_super_authority` is required to sign for a suitable replacement.

Set Reward Emissions Super Authority

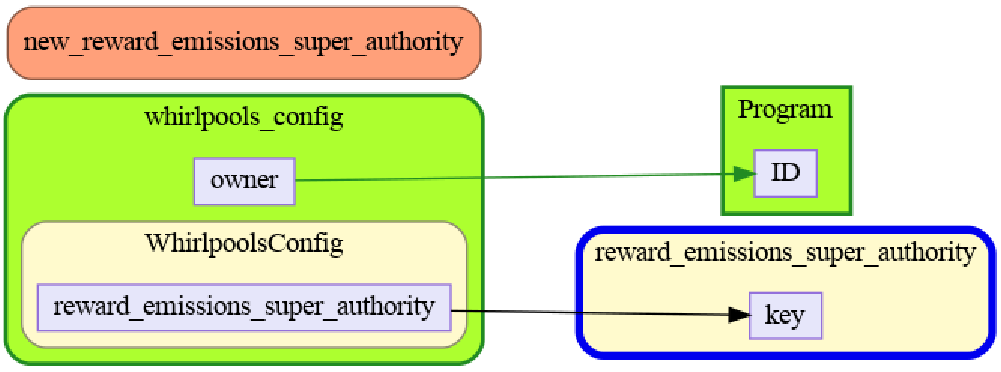
new_reward_emissions_super_authority
whirlpools_config
owner
WhirlpoolsConfig
reward_emissions_super_authority
Program
ID
reward_emissions_super_authority
key

Figure 24: Relationship graph for SetRewardEmissionsSuperAuthority instruction.

The `new_reward_emissions_super_authority` is passed as unchecked account input to replace the `reward_emissions_super_authority` of the `whirlpools_config`. The existing `reward_emissions_super_authority` is required to sign for a suitable replacement.

METHODOLOGY

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

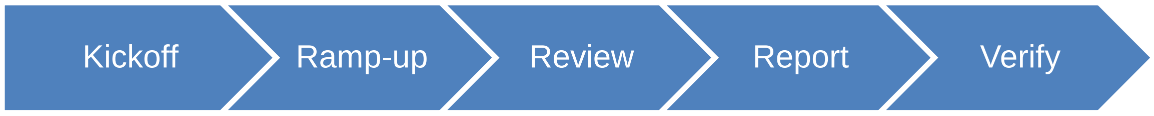

Figure 25: Methodology Flow.

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project and the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed 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 a 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 architecture analysis, a code review, 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 with the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the reviewer's experience. No dynamic testing was performed. Only 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 and not comprehensive, meant only to understand 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 PDF draft report containing an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement, including the number of findings and a statement about our general risk assessment of the project. 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 create a public report that can be shared and distributed to a larger audience.

Verify

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

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 and the agreement's scope.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This 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 – a 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 probability 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 overflow 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 nonstandard or non-peer-revied crypto functions
- Program crashes leaves core dumps or writes 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
Scott Carlson	Head of Blockchain	ScottJ.Carlson@kudelskisecurity.com