Architecture and Secure Code Assessment and Functionality Verification

Findings and Recommendations Report Presented to:

Axiom Markets Ltd.

April 14, 2022

Version: 3.0

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

Axiom Markets Ltd. engaged Kudelski Security to perform an Architecture and Secure Code Assessment and Functionality Verification.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on January 03 - March 09, 2022, 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.

A re-review was conducted on March 29, 2022.

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-HUB-01 – Prices not checked during Serum swap
- KS-HUB-02 – Missing authority checks for token accounts
- KS-HUB-03 – Pyth prices do not require a minimum of publishers
- KS-HUB-04 – Unchecked arithmetic
- KS-HUB-05 – Compiler Warnings - Some will become errors on future releases

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

At the time of re-review, we have verified that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

Kudelski performed an Architecture and Secure Code Assessment and Functionality Verification. 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://github.com/hubble-markets/hubble/ with the commit hash e45983e07f0e0cfa4792d3fb76b9aedf13c9e00b. The code was audited for a second time at the commit hash 40d45664e92342e55feb59b6f3d134cdf6cf7568.

Files included in the code review

```
hubble/
├── migrations/
│   └── deploy.js
├── programs/
│   ├── borrowing/
│   │   ├── data/
│   │   │   └── generate_positions.py
│   │   ├── src/
│   │   │   ├── borrowing_market/
│   │   │   │   ├── borrowing_operations.rs
│   │   │   │   ├── borrowing_rate.rs
│   │   │   │   ├── liquidation_calcs.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── tests_borrowing_config.rs
│   │   │   │   ├── tests_borrowing_multi_collateral.rs
│   │   │   │   ├── tests_borrowing_rate.rs
│   │   │   │   ├── tests_borrowing_single_collateral.rs
│   │   │   │   ├── tests_delegation.rs
│   │   │   │   ├── tests_deposit_and_borrow.rs
│   │   │   │   ├── tests_liquidation_calcs.rs
│   │   │   │   ├── tests_redistribution_multi_collateral.rs
│   │   │   │   ├── tests_redistribution_single_collateral.rs
│   │   │   │   ├── tests_utils.rs
│   │   │   │   └── types.rs
│   │   │   ├── config/
│   │   │   │   ├── global_config_operations.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── oracle_mapping_operations.rs
│   │   │   │   ├── tests_global_config_operations.rs
│   │   │   │   └── tests_oracle_mapping_operations.rs
│   │   │   ├── redemption/
│   │   │   │   ├── mod.rs
│   │   │   │   ├── redemption_operations.rs
│   │   │   │   ├── test_redemptions.rs
│   │   │   │   ├── test_utils.rs
│   │   │   │   └── types.rs
```

```
│   │   │   ├── stability_pool/
│   │   │   │   ├── liquidations_queue.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── stability_pool_operations.rs
│   │   │   │   ├── tests_hbb_issuance.rs
│   │   │   │   ├── tests_liquidations_queue.rs
│   │   │   │   ├── tests_stability_pool.rs
│   │   │   │   ├── tests_utils.rs
│   │   │   │   └── types.rs
│   │   │   ├── staking_pool/
│   │   │   │   ├── mod.rs
│   │   │   │   ├── staking_pool_operations.rs
│   │   │   │   ├── test_utils.rs
│   │   │   │   ├── tests.rs
│   │   │   │   └── types.rs
│   │   │   ├── state/
│   │   │   │   ├── borrowing_market_state.rs
│   │   │   │   ├── borrowing_vaults.rs
│   │   │   │   ├── collateral_amounts.rs
│   │   │   │   ├── deposit_snapshot.rs
│   │   │   │   ├── epoch_to_scale_to_sum.rs
│   │   │   │   ├── liquidations_queue.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── oracle_mappings.rs
│   │   │   │   ├── redemption_candidates_queue.rs
│   │   │   │   ├── redemptions_queue.rs
│   │   │   │   ├── stability_collateral_amounts.rs
│   │   │   │   ├── stability_pool_state.rs
│   │   │   │   ├── stability_provider_state.rs
│   │   │   │   ├── stability_token_map.rs
│   │   │   │   ├── stability_vaults.rs
│   │   │   │   ├── staking_pool_state.rs
│   │   │   │   ├── token_map.rs
│   │   │   │   └── user_staking_state.rs
│   │   │   ├── token_operations/
│   │   │   │   ├── hbb.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── soltoken.rs
│   │   │   │   ├── spltoken.rs
│   │   │   │   └── stablecoin.rs
│   │   │   ├── utils/
│   │   │   │   ├── bn.rs
│   │   │   │   ├── constraints.rs
│   │   │   │   ├── consts.rs
│   │   │   │   ├── coretypes.rs
│   │   │   │   ├── finance.rs
│   │   │   │   ├── macros.rs
```

```
│   │   │   │   ├── math.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── oracle.rs
│   │   │   │   ├── pda.rs
│   │   │   │   ├── test_utils.rs
│   │   │   │   └── tests_finance.rs
│   │   │   ├── borrowing_decoded_pubkey.txt
│   │   │   ├── handler_add_redemption_order.rs
│   │   │   ├── handler_approve_staking_pool.rs
│   │   │   ├── handler_approve_trove.rs
│   │   │   ├── handler_borrow_stablecoin.rs
│   │   │   ├── handler_clear_liquidation_gains.rs
│   │   │   ├── handler_clear_redemption_order.rs
│   │   │   ├── handler_delegate_deposit_marinade.rs
│   │   │   ├── handler_delegate_withdraw_marinade.rs
│   │   │   ├── handler_deposit_and_borrow.rs
│   │   │   ├── handler_deposit_collateral.rs
│   │   │   ├── handler_fill_redemption_order.rs
│   │   │   ├── handler_harvest_liquidation_gains.rs
│   │   │   ├── handler_harvest_staking_reward.rs
│   │   │   ├── handler_initialize_borrowing_market.rs
│   │   │   ├── handler_initialize_collateral_vaults.rs
│   │   │   ├── handler_initialize_stability_pool.rs
│   │   │   ├── handler_initialize_staking_pool.rs
│   │   │   ├── handler_repay_loan.rs
│   │   │   ├── handler_serum_close_account.rs
│   │   │   ├── handler_serum_init_account.rs
│   │   │   ├── handler_serum_swap.rs
│   │   │   ├── handler_stability_approve.rs
│   │   │   ├── handler_stability_provide.rs
│   │   │   ├── handler_stability_withdraw.rs
│   │   │   ├── handler_stake_hbb.rs
│   │   │   ├── handler_try_liquidate.rs
│   │   │   ├── handler_unstake_hbb.rs
│   │   │   ├── handler_update_global_config.rs
│   │   │   ├── handler_update_oracle_mapping.rs
│   │   │   ├── handler_withdraw_collateral.rs
│   │   │   └── lib.rs
│   │   ├── Cargo.toml
│   │   └── Xargo.toml
│   └── testwriter/
│       ├── src/
│       │   ├── handler_delegate_airdrop_marinade.rs
│       │   ├── lib.rs
│       │   └── testwriter_decoded_pubkey.txt
│       ├── Cargo.toml
│       └── Xargo.toml
```

```
├── src/
│   ├── client.ts
│   ├── config.ts
│   ├── data.ts
│   ├── delegation_constants.ts
│   ├── deployment.ts
│   ├── global.ts
│   ├── instructions_borrow.ts
│   ├── instructions_delegate.ts
│   ├── instructions_redeem.ts
│   ├── instructions_stability.ts
│   ├── instructions_staking.ts
│   ├── instructions_swap.ts
│   ├── main.ts
│   ├── pyth.ts
│   ├── set_up.ts
│   ├── utils.ts
│   ├── utils_display.ts
│   └── utils_serum.ts
├── Anchor.toml
├── Cargo.lock
├── Cargo.toml
├── Makefile
├── README.md
├── hubble-config.json
├── integration.Dockerfile
├── package-lock.json
├── package.json
├── tsconfig.json
├── tsts.txt
└── yarn.lock
```

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Architecture and Secure Code Assessment and Functionality Verification, we discovered:

- 1 finding with HIGH severity rating.
- 3 findings with MEDIUM severity rating.
- 1 finding with LOW severity rating.

The following chart displays the findings by severity.

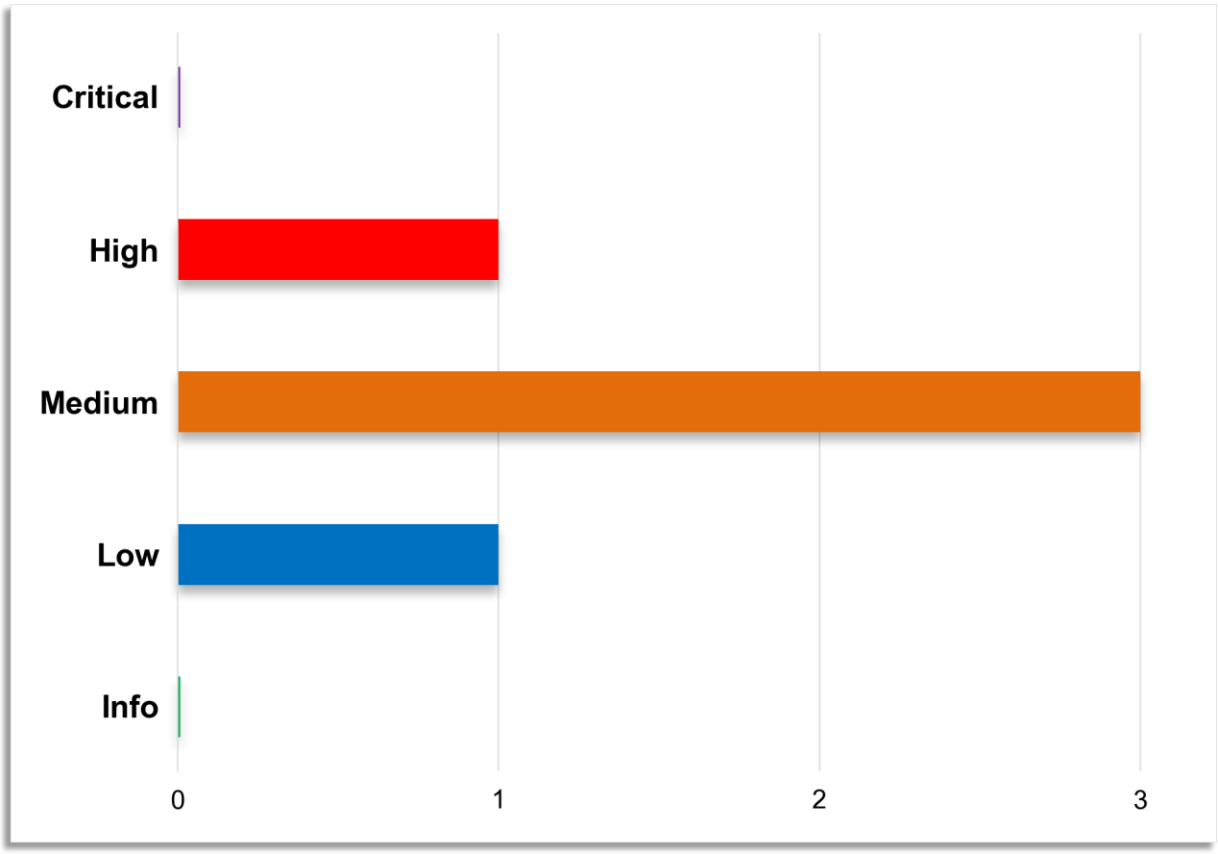

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each 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-HUB-01	**High**	Prices not checked during Serum swap
KS-HUB-02	**Medium**	Missing authority checks for token accounts
KS-HUB-03	**Medium**	Pyth prices do not require a minimum of publishers
KS-HUB-04	**Medium**	Unchecked arithmetic
KS-HUB-05	**Low**	Compiler Warnings - Some will become errors on future releases

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Further investigations concluded that no critical risks were identified for the application, including:

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

Conclusion

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

Technical Findings

General Observations

Hubble is a well-structured project with emphasis in testing. In fact, testing code coverage reaches 90%.

Using Anchor allows the project to impose account constraints quickly and intuitively. These provide integrity and increase the security of the program. There were only a couple of instances where these checks were omitted, listed in our findings below. These demonstrate how even small omissions can result to losses. Overall, however, the team seems to closely pay attention and understand the security checks that need to be performed by their programs.

Finally, the team was very willing and quick to provide us with the documentation needed, respond to our queries regarding the code and address disclosed issues.

Prices not checked during Serum swap

Finding ID: KS-HUB-01
Severity: **High**
Status: **Remediated**

Description

Pyth prices are not validated during in instruction `SerumSwapToUsdc`

Proof of Issue

File name: handler_serum_swap.rs
Line number: 1437

```
    // Oracle accounts
    pub pyth_sol_price_info: AccountInfo<'info>,
    pub pyth_eth_price_info: AccountInfo<'info>,
    pub pyth_btc_price_info: AccountInfo<'info>,
    pub pyth_srm_price_info: AccountInfo<'info>,
    pub pyth_ray_price_info: AccountInfo<'info>,
    pub pyth_ftt_price_info: AccountInfo<'info>,
    pub pyth_msol_price_info: AccountInfo<'info>,
```

Severity and Impact Summary

An attacker can select arbitrary oracle prices before placing an order in Serum DEX.

Recommendation

Match `pyth` accounts to the `oracle_mappings` of the borrowing market.

Missing authority checks for token accounts

Finding ID: KS-HUB-02
Severity: **Medium**
Status: **Remediated**

Description

Solana Token accounts are widely used in the project with their assets being handled by the program. The borrowing market, vaults and pools of the project are initialized with these accounts. Token accounts contain fields called

```
    pub delegate: COption<Pubkey>
    pub close_authority: COption<Pubkey>,
```

If these are set, the corresponding authorities could transfer tokens from the market's accounts.

Severity and Impact Summary

Incorrect initialization of parts of the market could lead to asset losses.

Recommendation

Authorities for token accounts managed by the program should be checked for a `None delegate` and `close_authority`. It is worth noting that Anchor's `TokenAccount` can be used since the project is already using the framework.

Remediation

Vault and fee accounts are now marked as `TokenAccounts` using Anchor, while the following constraints are added:

```
        constraint = burning_vault.delegate.is_none(),
        constraint = burning_vault.close_authority.is_none(),
```

Pyth prices do not require a minimum of publishers

Finding ID: KS-HUB-03
Severity: **Medium**
Status: **Open**

Description

> **Active Publishers**
>
> Pyth produces its price by aggregating price information from several different publishers. Each product in Pyth has a different number of publishers who are currently submitting prices. If a product has more publishers, each publisher has less influence on the overall price, which results in a more reliable aggregate price. If there are fewer than three publishers, then those publishers exert a substantial influence on the aggregate price.
>
> Before consuming a price feed, we suggest consulting the page for that product and determining how many publishers are actively submitting prices. For example, the BTC/USD product page is here: https://pyth.network/markets/#BTC/USD. If there are fewer than three publishers, we do not suggest using that price feed for high-value applications at the moment. We are working on adding more publishers to these products, and these feeds will become more robust over time.

- Pyth Best Practices

The number of publishers is not checked for pyth prices.

Severity and Impact Summary

As the Pyth documentation says:

> If there are fewer than three publishers, then those publishers exert a substantial influence on the aggregate price.

Recommendation

Although, the Pyth Best Practices just recommends to look at the Pyth market pages, the publishers can be calculated from Pyth price data by counting each unique `PriceComp::publisher` in a `Price::comp`. This should be performed both in initialization as well as when updating the oracle mapping.

References

- Pyth Best Practices

Unchecked arithmetic

Finding ID: KS-HUB-04
Severity: **Medium**
Status: **Remediated**

Description

Unchecked arithmetic is commonly missed and can result to integer flows, disrupting the correct function of programs. These can be easily pinpointed with:

```
cargo clippy -- -A clippy::all -W clippy::integer_arithmetic -W
clippy::integer_division
```

In Hubble's case, there are two notable cases:

Proof of Issue

File name: redemption_operations.rs
Line number: 800

```
    pub fn calculate_outstanding_redemption_amount(queue: &mut
RefMut<RedemptionsQueue>) -> u64 {
        queue
            .orders
            .iter()
            .map(|o| {
                if o.status != RedemptionOrderStatus::Inactive as u8 {
                    o.remaining_amount
                } else {
                    0
                }
            })
            .sum()
    }
```

File name: redemption_operations.rs
Line number: 64

```
    let outstanding_redemptions =
queue::calculate_outstanding_redemption_amount(queue);
    let remaining_supply = market.stablecoin_borrowed - outstanding_redemptions;
    if redemption_amount > remaining_supply {
        return Err(CannotRedeemMoreThanMinted);
    }
```

Where the sum of remaining amounts, in an extreme case, could overflow, while the remaining supply could be miscalculated if `outstanding_redemptions > market.stablecoin_borrowed`

Severity and Impact Summary

Unchecked arithmetic could result in program miscalculations in extreme cases.

Recommendation

Use Rust's `checked` or `saturating` arithmetic operations to catch or prevent errors.

Remediation

These portions of code have since been removed and are no longer in scope.

However, the linter additionally returns:

```
warning: integer arithmetic detected
   --> programs/borrowing/src/lib.rs:364:1
    |
364 | #[error]
    | ^^^^^^^^
```

Compiler Warnings - Some will become errors on future releases

Finding ID: KS-HUB-05
Severity: **Low**
Status: **Remediated**

Description

Building the program results in multiple warnings from `cargo`. These are summarized as follows:

- 9 warnings: borrow of packed field is unsafe and requires unsafe function or block (error E0133)
- 4 warnings: unused imports: `CloseAccount`, `self`
- 1 warning: unused import: `CandidateRedemptionUser`
- 1 warning: unused import: `CloseAccount`
- 1 warning: unused import: `spl_token::error::TokenError`
- 1 warning: unused imports: `AccountInfo`, `ProgramResult`, `msg`
- 1 warning: unused imports: `DEFAULT_DELEGATE_COLLATERAL_ALLOW_ACTIVE_ONLY`, `DEFAULT_TOTAL_DEBT_MAX`, `DEFAULT_USER_BORROW_MAX`, `DEFAULT_USER_BORROW_MIN`, `DEFAULT_USER_DEBT_MAX`, `DEFAULT_USER_DEBT_MIN`, `state::GlobalConfig`
- 1 warning: unused variable: `actual_gains_considering_precision_loss`
- 1 warning: unused variable: `debt`

While most of them refer to unused code, 9 warnings should be addressed, since according to `cargo`'s output:

```
warning: borrow of packed field is unsafe and requires unsafe function or block
(error E0133)
    ...
    = warning: this was previously accepted by the compiler but is being phased
out; it will become a hard error in a future release!
    = note: for more information, see issue #46043 <https://github.com/rust-
lang/rust/issues/46043>
    = note: fields of packed structs might be misaligned: dereferencing a
misaligned pointer or even just creating a misaligned reference is undefined
behavior
```

Proof of Issue

Building the project's programs with `./scripts/build.rs` runs successfully, reporting 20 warnings.

Severity and Impact Summary

Code incompatible with future compiler release.

Recommendation

Remove unused code and address E0133, preferably without introducing `unsafe` functionality.

References

- https://github.com/rust-lang/rust/issues/46043

Remediation

The project builds without any warnings.

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 off as 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. 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 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 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
Scott Carlson	**Blockchain Security**	**Scott.Carlson@kudelskisecurity.com**