Smart Contract Code and Secure Architecture Review

Findings and Recommendations Report Presented to:

Axiom Markets, LLC

March 02, 2022
Version: 1.0

Presented by:

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

STRICTLY CONFIDENTIAL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Axiom Markets, LLC engaged Kudelski Security to perform a Smart Contract Code and Secure Architecture Review.

The security evaluation was conducted remotely by the Kudelski Security Team. Testing took place on December 6th, 2021 –January 6th, 2022, and focused on the following objectives:

1. Provide the customer with an assessment of their overall security posture as it relates to their chosen architectural design and any risks that were discovered within the design of the system during the engagement.
2. To provide a professional opinion on the maturity, adequacy, and efficiency of smart contract code and the security measures that are in place.
3. To identify potential issues and include improvement recommendations based on the result of our analysis.

This report summarizes the engagement, component identification, and smart contract findings. It also contains detailed descriptions of the threats discovered and vulnerabilities as well as any applicable recommendations for remediation.

Key Findings

During the Smart Contract Code and Secure Architectural Review, the following themes were noted:

- There was some reliance on multiple programs and third-party incentives that contribute to stability in the application would benefit from more direct limitations (max variance) and financial security controls
- Components related to recovery and threshold collateral carry a tremendous amount of influence and trust that provides a considerable attack surface for a well-funded malicious actor
- Use of multi-step automated calculations results in precision loss and would result in severe impacts if compromised over fully decentralized applications
- Authentication and Access Control are high risk areas that should be tightly controlled through account privilege account management and instruction handling

Scope and Rules Of Engagement

Kudelski Security performed a Smart Contract Code and Secure Architecture Review for Axiom Markets LLC. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

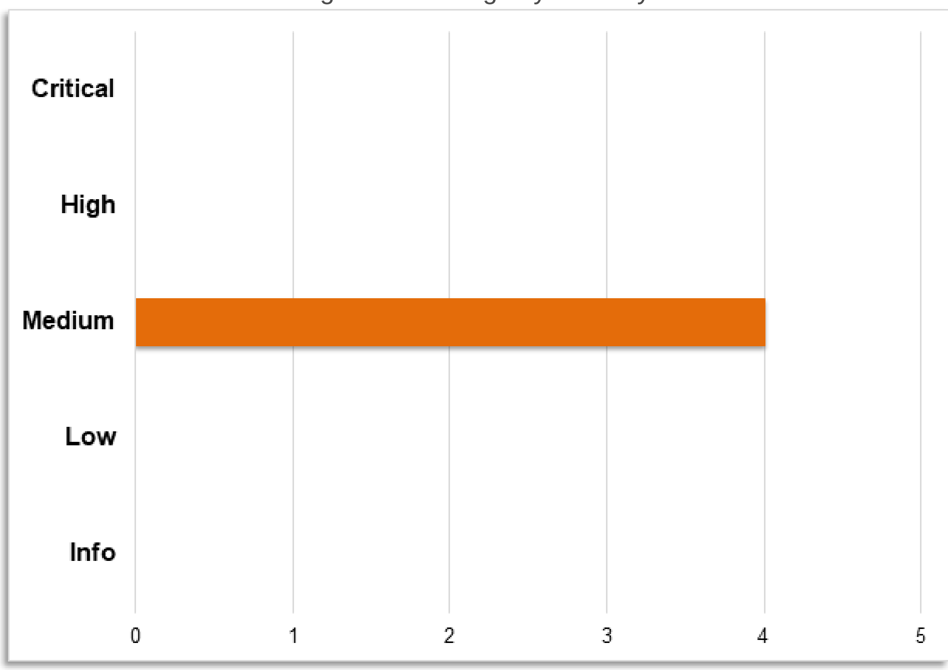
In-Scope Applications	
hubbleprotocol/hubble/	https://github.com/hubbleprotocol/hubble/releases/tag/0.1.0

TECHNICAL ANALYSIS & FINDINGS

During the Smart Contract Code Review, Kudelski Security discovered four (4) findings that had a medium 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.

FINDING ID	CLASSIFICATION	FINDING SEVERITY	DESCRIPTION	STATUS
1	Precision Loss in Stability Pool	**Medium**	Loss of precision occurs when calculating account values	Remediated
2	Precision Loss in Borrowing Operations	**Medium**	In recovery mode precision loss occurs when active collateral is being converted into inactive collateral	Remediated
3	Unsafe Arithmetic Operations - Overflow	**Medium**	In release (or optimization) mode, Rust silently ignores this by default and computes two's complement wrapping	Remediated
4	Unsafe Arithmetic Operations - Underflow	**Medium**	Instances in which calculation of number is more precise than is represented	Remediated

Table 1: Findings Overview

1 – Precision Loss in Stability Pool Key

Severity	**MEDIUM**
Score	5.3
CWE	CWE-1339: Insufficient Precision or Accuracy of a Real Number

Description: Loss of precision occurs when calculating account values where fractional components could result in loss or falling outside of the expected integral range.

Impact: Impact on the overall system is medium, but in certain circumstances has the possibility of being high. This is especially true in a system that changes constantly such as the Stability Pool, which is an essential component in the overall functionality and health of the financial eco-system.

Steps to Reproduce

```
294     stability_pool_state.cumulative_gains_total = stability_pool_state
295         .cumulative_gains_total
296         .add(&gains.to_token_map());
297     // TODO fix this when addressing precision loss for stability
298     // .add(&actual_gains_considering_precision_loss.to_token_map());
299     stability_pool_state.pending_collateral_gains = stability_pool_state
300         .pending_collateral_gains
301         .add(&gains.to_token_map());
302     // TODO fix this when addressing precision loss for stability
303     // .add(&actual_gains_considering_precision_loss.to_token_map());
304
```

Affected Resource
- stability_pool_operations.rs line 294
- stability_pool_operations.rs line 299

Recommendation: We recommend a fix to address the precision loss of the cumulation calculation by considering the precision loss that is likely to occur, which is dependent on volume and the factors of stability_pool_state as the project scales.

Reference
https://github.com/hubbleprotocol/hubble/blob/master/programs/borrowing/src/stability_pool/stability_pool_operations.rs#L297

Finding Status
Remediated code addresses the precision loss by the addition of actual_gains_considering_precision loss into the reward distribution calculation. The addition of Integer Overflow protections on the pool state mitigates the risk and does not pose a threat to calculation overflow/wraparound.

2 – Precision Loss in Borrowing Operations

Severity	MEDIUM
Score	4.7
CWE	CWE-1339: Insufficient Precision or Accuracy of a Real Number

Description - In recovery mode precision loss occurs when active collateral is being converted into inactive collateral and is triggered by withdraws (that can always be made in this specific application mode).

Impact - With varying levels of collateral at stake the impact of precision loss could be negligible but depending on the amount borrowed it could prove to be more severe especially if the stablecoin at that moment is not perfectly pegged at it 1:1 target.

Steps to Reproduce

```
217         market.stablecoin_borrowed = market
218             .stablecoin_borrowed
219             .checked_sub(payment_amount)
220             .unwrap();
221         user.borrowed_stablecoin = updated_stablecoin_borrowed;
222
223         redistribution::update_user_stake_and_total_stakes(market, user);
224
225         if updated_stablecoin_borrowed == 0 {
226             // If after repayment it's empty, then we don't consider this an active loan
227             user.status = UserStatus::Inactive as u8;
228             market.num_active_users -= 1;
229
230             // Turn collateral to inactive
231             market
232                 .deposited_collateral
233                 .sub_assign(&user.deposited_collateral);
234             market
235                 .inactive_collateral
236                 .add_assign(&user.deposited_collateral);
237             user.inactive_collateral
238                 .add_assign(&user.deposited_collateral);
239             user.deposited_collateral = CollateralAmounts::default();
240         }
241
242         Ok(RepayLoanEffects {
243             amount_to_burn: payment_amount,
244             amount_to_transfer: payment_amount,
245         })
246     }
```

Affected Resource

- borrowing_operations.rs line 225

Recommendation -We recommend a fix to address precision loss that occurs in recovery mode by considering the precision loss that is likely to occur, which is dependent on volume and the factors such as the max withdraws allowed in recovery mode.

Reference

- https://github.com/hubbleprotocol/hubble/blob/master/programs/borrowing/src/borrowing_market/borrowing_operations.rs#L225

Finding Status – The precision errors that were noted in the findings have been addressed by changing the checks and structure of *update_stablecoin_borrowed* in which the updated code makes use of a *.user_debt_min* constraint. The system features that take into account the updating of user balances in a redistribution event and the conversion to inactive collateral appear to function properly and follow best coding practices.

3 – Unsafe Arithmetic Operations – Overflow

Severity	**MEDIUM**
Score	5.9
CWE	CWE-190: Integer Overflow or Wraparound

Description

Rust includes checks for integer overflow and underflow that cause your program to panic at runtime if this behavior occurs. **However, in release (or optimization) mode, Rust silently ignores this by default and computes two's complement wrapping**. In other words, an overflow or underflow in Rust that appears as panic when debugging may disappear when deployed in production.

Steps to Reproduce

Overflow

There were 5 instances of code that could lead to overflow.

1. **Location:** line 71, column 18 in programs/borrowing/src/utils/finance.rs

   Description: It is unlikely that the mv variable will ever overflow, due to the insane market value that would represent. However, using checked_add would still prevent this issue, regardless of its likeliness of being exploited.

```
61
62      pub fn calc_market_value_usdh(prices: &TokenPrices, amounts: &CollateralAmounts) -> u64 {
63          use CollateralToken::*;
64          let sol = Self::calc_market_value_token(amounts.sol, &prices.sol, SOL);
65          let eth = Self::calc_market_value_token(amounts.eth, &prices.eth, ETH);
66          let btc = Self::calc_market_value_token(amounts.btc, &prices.btc, BTC);
67          let srm = Self::calc_market_value_token(amounts.srm, &prices.srm, SRM);
68          let ray = Self::calc_market_value_token(amounts.ray, &prices.ray, RAY);
69          let ftt = Self::calc_market_value_token(amounts.ftt, &prices.ftt, FTT);
70          let msol = Self::calc_market_value_token(amounts.msol, &prices.msol, MSOL);
71          let mv = sol + eth + btc + srm + ray + ftt + msol;
72          mv as u64
73      }
74
```

2. **Location:** line 307 in programs/borrowing/src/stability_pool/stability_pool_operations.rs

   Description: The argument provided to the checked_sub() method uses raw addition, which is susceptible to overflow. It is possible to use nested checked_* arithmetic, in this case, a checked_add() could be used within the checked_sub() method call.

```
    stability_pool_state.stablecoin_deposited = stability_pool_state
        .stablecoin_deposited
        .checked_sub(
            usd_loss
                + last_usd_error
                    .checked_div(DECIMAL_PRECISION as u64)
                    .unwrap(),
        )
        .unwrap();

    Ok(())
}
```

3. **Location:** lines 209, 213, 218, 223, 224 in programs/borrowing/src/stability_pool/liquidations_queue.rs

   Description: Multiple instances of raw addition in drain_event! macro definition

```
208
209                 $stability_pool_gains.$token += $event.collateral_gain_to_stability_pool.$token;
210                 $event.collateral_gain_to_stability_pool.$token = 0;
211
212                 // What belongs to the clearer no matter what
213                 $clearing_agent_gains.$token += $event.collateral_gain_to_clearer.$token;
214                 $event.collateral_gain_to_clearer.$token = 0;
215
216                 // What belongs to the liquidator
217                 if $clearing_agent_is_event_liquidator {
218                     $clearing_agent_gains.$token += $event.collateral_gain_to_liquidator.$token;
219                     $event.collateral_gain_to_liquidator.$token = 0;
220                 }
221
222                 // What belongs to the clearer if the liquidator is lazy
223                 if $event.event_ts + LIQUIDATIONS_SECONDS_TO_CLAIM_GAINS < $now {
224                     $clearing_agent_gains.$token += $event.collateral_gain_to_liquidator.$token;
225                     $event.collateral_gain_to_liquidator.$token = 0;
226                 }
227             }};
228         }
229     }
```

Affected Resources

- programs/borrowing/src/state/mod.rs
- programs/borrowing/src/borrowing_market/liquidation_calcs.rs
- programs/borrowing/src/stability_pool/liquidations_queue.rs
- programs/borrowing/src/borrowing_market/borrowing_operations.rs
- programs/borrowing/src/stability_pool/stability_pool_operations.rs

<u>**Recommendation:**</u> It is **recommended to use checked_add, checked_sub, checked_mul, checked_div, and checked_pow** instead of using raw arithmetic.

Finding Status

All locations in which unsafe arithmetic operations were a threat have been mitigated with the addition of the recommended checks including *.checked_add* on the market value calculation of the collateral tokens, that are supported by the platform. The addition of *.checked_add* in the reworking of the *mv* =sol variable when calculating the stability pool state was also mitigated by using thing recommended addition check.

4 – Unsafe Arithmetic Operations – Underflow

Severity	**MEDIUM**
Score	5.9
CWE	CWE-191: Integer Underflow (Wrap or Wraparound)

Underflow

There were 5 instances of code that could lead to underflow.

1. **Location**: line 648, column 13 in programs/borrowing/src/borrowing_market/borrowing_operations.rs

   Description: Although it is hard to imagine a scenario in which this could be exploited, it is still recommended that checked_sub() be used when calculating "diff_stablecoin_reward_per_token" as extra precaution.

```
639
640     pub fn get_pending_redistributed_stablecoin_reward(
641         market: &BorrowingMarketState,
642         user: &UserMetadata,
643     ) -> u64 {
644         let snapshot_stablecoin_reward_per_token = user.user_stablecoin_reward_per_token;
645
646         let latest_stablecoin_reward_per_token = market.stablecoin_reward_per_token;
647         let diff_stablecoin_reward_per_token =
648             latest_stablecoin_reward_per_token - snapshot_stablecoin_reward_per_token;
649
650         if diff_stablecoin_reward_per_token == 0 || user.status != (UserStatus::Active as u8) {
651             return 0;
652         }
```

2. **Location:** line 189, column 30 in programs/borrowing/src/borrowing_market/borrowing_operations.rs

   Description: Although it requires the unusual condition of borrowing fees being larger than the amount_to_borrow itself, this could have devastating effects when calculating the amount of tokens to mint to a user. It is recommended that checked_sub() be used, and possibly institute a check for the unusual situation where fees would be larger than the amount_to_borrow.

```
187
188     Ok(BorrowStablecoinEffects {
189         amount_mint_to_user: borrow_and_fee.amount_to_borrow - borrow_and_fee.fees_to_pay,
190         amount_mint_to_fees_vault: staking_fee,
191         amount_mint_to_treasury_vault: treasury_fee,
192     })
193 }
```

3. **Location:** line 748, column 26 in programs/borrowing/src/stability_pool/stability_pool_operations.rs

   Description: Possible underflow in scale_diff definition.

```
            return 0;
        }

        let scale_diff = stability_pool_state_current_scale - scale_snapshot;

        (match scale_diff {
            0 => (initial_stake as u128)
                .checked_mul(stability_pool_state_p)
```

4. **Location:** line 65, column 28 in programs/borrowing/src/redemption/redemption_operations.rs

   Description: Possible underflow in remaining_supply definition

```
62      }
63
64      let outstanding_redemptions = queue::calculate_outstanding_redemption_amount(queue);
65      let remaining_supply = market.stablecoin_borrowed - outstanding_redemptions;
66      if redemption_amount > remaining_supply {
67          return Err(CannotRedeemMoreThanMinted);
68      }
69
70      if now_timestamp < market.bootstrap_period_timestamp {
```

Affected Resources

1. line 648, column 13 in programs/borrowing/src/borrowing_market/borrowing_operations.rs
2. at line 189, column 30 in programs/borrowing/src/borrowing_market/borrowing_operations.rs
3. line 258 in programs/borrowing/src/borrowing_market/borrowing_operations.rs
4. at line 63, column 28 in programs/borrowing/src/redemption/redemption_operations.rs

Recommendation

It is **recommended to use checked_add, checked_sub, checked_mul, checked_div, and checked_pow** instead of using raw arithmetic.

Finding Status

All four locations in which unsafe arithmetic operations were part of the findings have been mitigated. The *get_pending_redistributed_collateral_reward* function now follows security best practices. The BorrowStableEffects finding was also remediated through the removal of the calculation and is replaced by the *requested_borrow_amount*. The underflow issues that remained in the final two instances have been remediated with the use of `.checked_sub`.

Additional Notes – Architecture Review

While validating that the above vulnerabilities had been patched, Kudelski Security would also like to validate the additions made after our initial review and clarify any points of question or concern. As we conducted the architectural review before the smart contract code review findings were described based in general terms based around the higher-level architecture.

Authentication and Access Control was a general finding describing the potentials of compromise in the Solana ecosystem through the assumption that one does not need to explicitly check the owner admin-only instructions. Specific points in the codebase were not identified but we do recommend in any updates to diligently check the owner of an account before interactions with a (AccountInfo::owner field).

Malicious Input Handling was a general finding we mention in many Solana projects that without the account ownership and instruction checks that account handling this allows for the state data to be influenced or corrupted by the input of malicious accounts recommend that program_ids that are provided are checked against what the smart contract program reasonably expects so that the malicious accounts won't have the capacity to provide unexpected input

Tools

The precompile code review process is primarily manual, but the following tools were used during this assessment to rule out common vulnerabilities and weaknesses that may not have been captured during the software development lifecycle.

- Manual Analysis Tools
  - Microsoft Visual Studio Code
  - IntellJ IDEA
  - Soteria

Vulnerability Scoring Systems

Our Application Security Team utilizes three common vulnerability scoring systems to assign a risk severity to findings.

- Common Vulnerability Scoring System (CVSS)
- Common Weakness Enumeration (CWE)

In this document, CVSS, along with CWE, is used for network-based evaluations, and the OWASP Top 10, along with the CWE, is used for web applications. Using these metrics, each vulnerability identified during testing is classified by severity.

CVSS

The CVSS 3.1 scoring system provides a Base Score arrived at by objective measures, which can be modified by a Temporal Score and Environmental Score, which take other factors into consideration. The CVSS base score assigns severity to vulnerabilities found. The Base Score with the Temporal and Environmental scores allow the Application Security Team to prioritize responses and resources according to the threat. Scores range from 0 to 10, with 10 being the most severe.

CVSS Base and Temporal scores are represented as a numeric value and also as a vector string. The vector string is a textual representation of the metric values used to determine the score.

CVSS 3.1 Calculator, provided by NIST: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

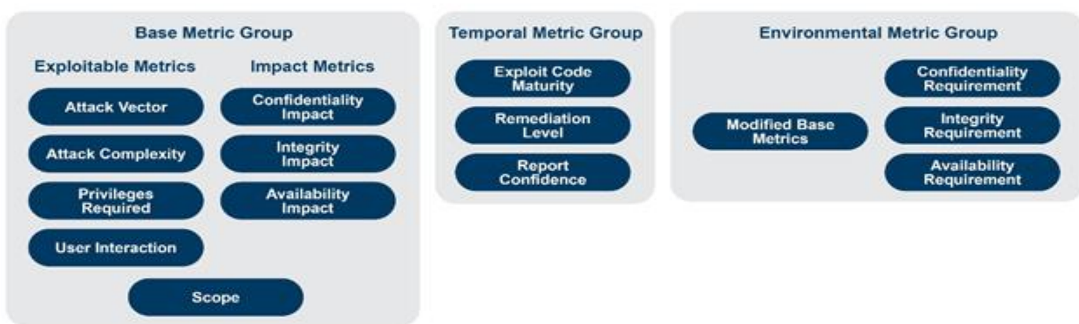


Figure 2: CVSS Scoring System

CVSS 3.1 Ratings	
Critical	9.0 to 10.0
High	7.0 to 8.9
Medium	4.0 to 6.9
Low	0.1 to 3.9
None	0.0

Table 2: CVSS Ratings

CWE

The CWE system is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts. Some common types of software weaknesses classified by the CWE are:

- Buffer Overflows, Format Strings, etc.
- Structure and Validity Problems
- Common Special Element Manipulations
- Channel and Path Errors
- Handler Errors
- User Interface Errors
- Pathname Traversal and Equivalence Errors
- Authentication Errors
- Resource Management Errors
- Insufficient Verification of Data
- Code Evaluation and Injection
- Randomness and Predictability

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Shannon Garcia	Practice Lead - Application Security & Threat Management	Shannon.Garcia@kudelskisecurity.com +1-512-516-5413
Brandon Gilchrist	Senior Blockchain Expert	brandon.gilchrist@kudelskisecurity.com
Michael Erquitt	Senior Blockchain Analyst	michael.erquitt@kudelskisecurity.com
Brian Olson	Senior Blockchain Analyst	brian.olson@kudelskisecurity.com