Kujira - USK Stablecoin Contracts - Audit Report

Prepared for Kujira, 6 September 2022

Table of Contents

Introduction

SCV was engaged by Kujira to assist in identifying security threats and vulnerabilities that have the potential to affect their security posture. Additionally, SCV will assist the team in understanding the risks and identifying potential mitigations.

Scope

SCV performed the security assessment on the following codebase:

- https://github.com/Team-Kujira/stable
- Code Freeze: *79d05b0d5b16e1f204f349787d16649754e6c5eb* (Tag v1.1.0)

The complementary Proof of Concept (PoC) test cases mentioned along vulnerabilities findings can be accessed via the link below:

- https://gist.github.com/scvsecurity/48531c2e2aabd63f301f92327174331a

SCV notes that, financial attacks were not part of the scope.

Remediations were applied into several commits up to the following hash commit:

- Code Freeze *b4f113edaa8ee3c05e0baa3b1a050526b77f22ef*

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to Kujira. Testing includes, but is not limited to, the following:

- Understanding the application and its code base purpose;
- Deploying SCV in-house tooling to automate dependency analysis and static code review;
- Analyse each line of the code base and inspect application security perimeter;
- Review underlying infrastructure technologies and supply chain security posture;

Code Criteria and Test Coverage

SCV used a scale from **0** to **10** that represents how **SUFFICIENT(6-10)** or **NOT SUFFICIENT(0-5)** each code criteria was during the assessment:

Criteria	Status	Scale Range	Notes
Provided Documentation	**Sufficient**	7-8	N/A
Code Coverage Test	**Sufficient**	7-8	N/A
Code Readability	**Sufficient**	6-8	N/A
Code Complexity	**Sufficient**	7-8	N/A

Vulnerabilities Summary

	Title and Summary	Risk	Status
1	Margin contract allows users to extract risk-free value from the margin position	**Critical**	**Remediated**
2	Permissionless function allows unauthorized callers to change configuration parameters	**Critical**	**Remediated**
3	Automatic liquidations will fail if there is an insolvent position	**High**	**Remediated**
4	Insolvent loans can only be liquidated by sudo execution that can cause bad debt to be incurred due to the governance voting process	**Medium**	**Remediated**
5	Sudo execution allows liquidating an overcollateralized position	**Medium**	**Remediated**
6	Lack of validation during contract instantiation and update can lead to misconfigurations.	**Low**	**Remediated**
7	Opening margin position does not incur minting fees nor is checked against max debt	**Low**	**Remediated**
8	Updating interest rate might introduce an unfair fee-charging mechanism	**Low**	**Remediated**
9	Consider checking the permitted address to be unique to prevent duplicate addresses	**Informational**	**Remediated**
10	Consider returning an informational error if the user partial liquidated too much collateral	**Informational**	**Acknowledged**

Detailed Vulnerabilities

1. Margin contract allows users to extract risk-free value from the margin position

Likelihood	Impact	Risk
Likely	Critical	**Critical**

Description

The `execute_open` function in `stable/contracts/margin/src/contract.rs:113` allows users to open a margin USK position. Unlike the traditional stable market contract, the margin contract does not require up-front collateral for a user to open a position. This is problematic because the margin contract shares the same core functions as the traditional market contract, most importantly, the mint and the liquidation functions. These functions have not been adapted to reflect that the margin position owner does not own the initial `deposit_amount` and effectively has no vested interest in the position.

Due to this, a user may be able to extract risk-free value from the protocol. This can be accomplished in two main ways. Firstly, by calling `Mint` on their open margin position to mint additional USK within a healthy LTV. The user is able to receive minted USK for a position where they have not posted any collateral of their own. So in effect, the user has no obligation to repay this debt. This issue is also present in liquidations. A user may self-liquidate their position which will in effect send the initial collateral created during the margin position creation.

This is possible because, during the creation of the position, the user is not required to post collateral of any sort.

Please see the test cases `test_steal_contract_funds` and `test_steal_mint_funds` for proof of concept link below:

- https://gist.github.com/scvsecurity/48531c2e2aabd63f301f92327174331a#file-kujira-usk-stable-poc-rs-L408
- https://gist.github.com/scvsecurity/48531c2e2aabd63f301f92327174331a#file-kujira-usk-stable-poc-rs-L487

Recommendations

We recommend re-architecting the margin contract to enforce an amount of collateral to be posted before a user can enter a margin position. We recommend removing `mint_amount` and `swap_amount` from `execute_open` which should be replaced with functionality to determine the positions collateral from `info.funds`. In addition to adding this functionality to the `execute_open`, the margin contract also needs custom mint, burn, and liquidation functionality to ensure that these operations are being performed correctly given that the position is not a traditional position. Additionally, it may be beneficial to create a `MarginPosition` struct to better handle the parameters associated with a margin position.

2. Permissionless function allows unauthorized callers to change configuration parameters

Likelihood	Impact	Risk
Likely	Critical	**Critical**

Description

In the margin and market contracts, there is no check whether the sender is the contract owner when updating configuration parameters. As a result, an attacker can update the `max_ratio` of the market config to a very high value, such as `Decimal::from_ratio(1000u128, 1u128)`, which allows anyone to borrow a large number of stables with little collateral values due to *100_000%* LTV ratio. This would result in a loss of funds for the protocol and end up being insolvent.

Affected code lines:

- `stable/contracts/margin/src/contract.rs:97`
- `stable/contracts/market/src/contract.rs:64`

Recommendations

When updating the contract's configuration, consider validating that the caller is the contract's owner.

3. Automatic liquidations will fail if there is an insolvent position

Likelihood	Impact	Risk
Possible	Severe	**High**

Description

The `execute_liquidates` function in `stable/packages/stable/src/contract.rs:217` allows the liquidator to perform automatic or manual liquidation with a given addresses. Since the liquidation of insolvent functions is a privileged function that can only be performed through sudo execution, automatic liquidation would fail in line 239 if any liquidatable addresses are insolvent.

The trader must filter out insolvent positions and perform manual liquidations to overcome this. However, the `Liquidatable` query that returns liquidatable addresses does not filter insolvent positions. This forces the third-party liquidator to remove insolvent positions manually after querying liquidatable positions.

As a result, liquidations are delayed, and positions that need to be liquidated immediately are not liquidated, causing them to become insolvent and potentially accrue bad debt to the protocol. This might cause catastrophic damage to the protocol's solvency during a market crash.

Please see the `test_automatic_liquidation_insolvent_position` test case for proof of concept link below:

- https://gist.github.com/scvsecurity/48531c2e2aabd63f301f92327174331a#file-kujira-usk-stable-poc-rs-L249

Recommendations

Consider reworking the automatic liquidation system to exclude insolvent positions so third-party liquidators can perform in-time liquidations during a market crash. Additionally, consider adding a new liquidation query message that excludes insolvent positions so third-party liquidators can query solvent positions and perform manual liquidation towards them.

4. Insolvent loans can only be liquidated by sudo execution that can cause bad debt to be incurred due to the governance voting process

Likelihood	Impact	Risk
Unlikely	Severe	**Medium**

Notes

The team resolved this issue by removing the sudo execution entry point and instead allowing the option for third parties (such as governance) to re-collateralize any insolvent positions to the point that liquidators can liquidate them as usual.

Description

In `stable/packages/stable/src/contract.rs:380`, insolvent loans can only be liquidated by sudo execution.

Sudo execution can only be executed from governance which requires time to vote and execute. This means that the longer the time taken for the proposal to be voted and executed, the more bad debt the protocol incurs if the price of collateral keeps dropping.

We recognize that it is a design choice to handle insolvent liquidations via governance to avoid actualizing protocol losses, but it opens the possibility to where a collateral asset may continue dropping further in price during the governance process. Even an accelerated governance process may cause a delay to execute liquidations on a position-by-position basis. Ultimately this is a design decision for an edge case that is highly unlikely to occur unless liquidations are blocked or a collateral asset has a sharp decrease in price similar to Luna Classic in May 2022.

Recommendations

Rather than propose code fixes, we will propose design considerations. We recommend strictly outlining how insolvent loans will be handled by governance. The current sudo implementation is sufficient to protect the protocol against creating bad debt by liquidating positions during the event of a sharp wick down followed by a return in price above the liquidation price. This comes at the cost of quick action in the case that a collateral is experiencing a sharp continued decline (Luna Classic). One way to combat this situation is to add a sudo execution that may allow for an "emergency mode" to allow liquidations for a specified collateral's insolvent loans to be liquidated rather than having to perform a sudo liquidation for each position id.

5. Sudo execution allows liquidating an overcollateralized position

Likelihood	Impact	Risk
Unlikely	Moderate	**Medium**

Notes

The team resolved this issue along with issue 4 by modifying the liquidation mechanism for third parties to re-collateralize insolvent positions.

Description

In `stable/packages/stable/src/contract.rs:380-404`, the sudo liquidation does not check whether the position is overcollateralized before liquidating. This means that if a safe position is incorrectly included during sudo liquidation, their position will get liquidated even though their position is safely overcollateralized.

Please see the `test_sudo_liquidate_safe_position` test case for proof of concept link below:

- https://gist.github.com/scvsecurity/48531c2e2aabd63f301f92327174331a#file-kujira-usk-stable-poc-rs-L2

Recommendations

Consider ignoring overcollateralized positions when performing sudo liquidation.

6. Lack of validation during contract instantiation and update can lead to misconfigurations.

Likelihood	Impact	Risk
Possible	Low	**Low**

Description

When performing contract instantiation and updating the configuration, there are no validations being performed on the message parameters.

Ideally, both `Addr` and `Decimal` values should be validated to ensure that no invalid addresses are being saved into storage and overflow issues happening when decimals are being used for calculation.

Affected code lines and variables:

- `stable/contracts/margin/src/contract.rs:29`
  - `fin_address`, `owner`, `stable_denom_admin`, `orca_address`, `max_ratio`, `mint_fee`, `interest_rate`, `liquidation_ratio`
- `stable/contracts/margin/src/contract.rs:97`
  - `fin_address`, `owner`, `orca_address` `max_ratio`, `mint_fee`, `interest_rate`, `liquidation_ratio`
- `stable/contracts/market/src/contract.rs:33`
  - `fin_address`, `owner`, `stable_denom_admin`, `orca_address`, `max_ratio`, `mint_fee`, `interest_rate`, `liquidation_ratio`
- `stable/contracts/market/src/contract.rs:64`
  - `fin_address`, `owner`, `orca_address`, `max_ratio`, `mint_fee`, `interest_rate`, `liquidation_ratio`
- `stable/contracts/mint/src/contract.rs:25`
  - `owner`
- `stable/contracts/mint/src/contract.rs:69`
  - `address`

Recommendations

- Validate the `Addr` string before saving it into storage
- Ensure all `Decimal` values are lower or equal to `Decimal::one()`

7. Opening margin position does not incur minting fees nor is checked against max debt

Likelihood	Impact	Risk
Possible	Low	**Low**

Description

When opening a margin position in `stable/contracts/margin/src/contract.rs:113`, there is no validation that checks the new mint total is lesser than `config.max_debt` and no calculation of mint fee taken from the position. The former would cause the admin to be unable to limit the max minted amount in the contract, while the latter would cause the protocol not to get any revenue when opening a position.

Note that checking against max debt and charging a minting fee is done during `execute_mint` as seen in `stable/packages/stable/src/contract.rs:125-147`.

Recommendations

Consider validating the new mint total is lesser than `config.max_debt` and charging a minting fee when opening a margin position.

8. Updating interest rate might introduce an unfair fee-charging mechanism

Likelihood	Impact	Risk
Possible	Low	**Low**

Description

In `stable/packages/stable/src/position.rs:446`, the `update_interest` function would increase the user's interest amount based on the admin configured interest rate. Since the interest amount is only updated when the user performs any operation that updates the interest rate such as depositing extra collateral, a new interest rate configured by the admin might cause inconsistent interest rates among all users and introduce an unfair fee-charging mechanism.

Please see the `test_interest_arbitrary_update` test case for proof of concept link below:

- https://gist.github.com/scvsecurity/48531c2e2aabd63f301f92327174331a#file-kujira-usk-stable-poc-rs-L92

Recommendations

Consider reworking the `update_interest` function to correctly accrue the user's position interest amount based on the elapsed period.

9. Consider checking the permitted address to be unique to prevent duplicate addresses

Likelihood	Impact	Risk
Rare	Informational	**Informational**

Description

In `stable/contracts/mint/src/contract.rs:69`, the contract owner can permit the same address into the `permitted` vector. This is inefficient as having duplicate addresses is not needed and should be avoided.

Recommendations

Consider checking the address to permit does not exist in the `permitted` vector to avoid duplicate addresses.

10. Consider returning an informational error if the user partial liquidated too much collateral

Likelihood	Impact	Risk
Possible	Informational	**Informational**

Notes

The team mentioned that there's no real need to constrain the liquidation amount at the contract level as the front end can calculate the amount needed for partial liquidation.

Description

When performing a partial liquidation `stable/packages/stable/src/contract.rs:200`, the depositor can supply an arbitrary `amount` to self-liquidate some of the collateral to repay their loan. If the user accidentally supplied too much collateral to repay, an overflow error would occur in `stable/packages/stable/src/position.rs:297`. Since the actual error that happened is because the user-provided too much collateral to partially liquidate, the overflow error would confuse the user and will affect the user experience when interacting with the protocol.

Recommendations

Consider checking if repay amount is bigger than `self.mint_amount`, if yes, return an informative error saying too much collateral is provided to liquidate minted stables.

Document control

Document changes

Version	Date	Name	Changes
0.1	2022-08-28	Vinicius Marino	Initial report
0.2	2022-08-31	Vinicius Marino	Team communication and Pre-Release
1.0	2022-09-06	Vinicius Marino	Revisions and Document Release

Document contributors

Name	Role	Email address
Vinicius Marino	Security Specialist	vini@scv.services

Appendices

Appendix A: Report Disclaimer

Appendix B: Risk assessment methodology

A qualitative risk assessment is performed on each vulnerability to determine the impact and likelihood of each.

Risk rate will be calculated on a scale. As per criteria Likelihood vs Impact table below:

Impact \ Likelihood	Rare	Unlikely	Possible	Likely
Critical	**Medium**	**High**	**Critical**	**Critical**
Severe	**Low**	**Medium**	**High**	**High**
Moderate	**Low**	**Medium**	**Medium**	**High**
Low	**Low**	**Low**	**Low**	**Medium**
Informational	**Informational**	**Informational**	**Informational**	**Informational**

LIKELIHOOD:

- **Likely**: likely a security incident will occur;
- **Possible**: It is possible a security incident can occur;
- **Unlikely**: Low probability a security incident will occur;
- **Rare**: In rare situations, a security incident can occur;

IMPACT:

- **Critical**: May cause a significant and critical impact;
- **Severe**: May cause a severe impact;
- **Moderate**: May cause a moderated impact;
- **Low**: May cause low or none impact;
- **Informational**: May cause very low impact or none.

SCV