AUDIT REPORT

XionMarkets

Markets Contracts

Prepared by SCV-Security

On 2nd March 2025

Table of Contents

Introduction

SCV has been engaged by XionMarkets to conduct a comprehensive security review with the goal of identifying potential security threats and vulnerabilities within the codebase. The purpose of this audit is to evaluate the security posture of the codebase and provide actionable recommendations to mitigate any identified risks. This report presents an overview of the findings from our security audit, outlining areas of concern and proposing effective measures to enhance the codebase's security.

Scope Functionality

The XionMarkets implement the factory and market contracts. The factory contract serves as a proxy for constructing and administering market contracts, which function as Automated Market Makers (AMMs) for forecasts. Users can only engage with market contracts through the factory contract, which is responsible for executing calls to these markets.

Submitted Codebase

XionMarkets Contracts	
Repository	https://github.com/EdinyangaOttoho/xionmarkets
Commit	`dc2ba071b4c24fc99087cbf39f0d7af7dc0f9493`
Contracts	`factory` & `market`
Branch	`master`

Remediations – Submitted Codebase

XionMarkets Contracts	
Repository	https://github.com/EdinyangaOttoho/xionmarkets
Commit	13cede35889560631bf5a9b8fced4346f255e343
Contracts	factory & market
Branch	master

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 XionMarkets. Testing includes, but is not limited to, the following:

- Understanding the application and its functionality 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.

Security Assessment Questionnaire

In addition to the above methodologies, SCV conducted a targeted (Xion's app-chain context) questionnaire-based assessment to address key security concerns, ensuring a thorough evaluation of potential risks. The following aspects were specifically examined:

1. **Administrator Privileges & Fund Security**
   - 1.1. Do contract administrators or owners have privileged functions that could negatively impact users, such as the ability to steal or block funds?
   - 1.2. Are administrative controls appropriately restricted to minimize security risks while maintaining necessary functionality?

2. **Mitigation of Fee Drain via Spam Vectors**
   - 2.1. Could certain contract functions be exploited to drain fee funds, particularly within the Xion fee-sponsoring model?
   - 2.2. Do execution functions include self-limiting mechanisms, such as cost implications or configurable constraints, to prevent spam-based depletion of resources?

Security Assessment Results

1. Administrator Privileges & Fund Security

The user funds are held in the market contracts, which are instantiated by the factory contract. However, in `contracts/factory/execute.rs:75`, the market contract migration admin is set to the factory contract.

This means that the factory contract's migration admin can perform a contract migration for the market contract. If the admin is compromised, the attacker can migrate the factory contract to overwrite the code ID and steal funds from the market contract.

Other than that, the contract does not give administrators direct control over user funds. Privileged functions are restricted to maintenance operations and secured via role-based access controls. Administrators should take general precautions when interacting with the protocol, such as being cautious when resolving markets to prevent selecting the wrong variant.

2. Mitigation of Fee Drain via Spam Vectors

The factory contract performs calls to the market contracts. However, these markets are not validated to be the markets instantiated by the factory contract.

For example, an attacker can upload and instantiate a dummy contract that does nothing when the `ExecuteMarketMsg:RemoveLiquidity` message is called. The attacker can then call the `RemoveLiquidity` message in the factory contract with the `market` address pointing to their contract address. This would cause empty executions to occur, wasting fee funds. With that said, since the Xion chain is permissioned, the attacker must find a way to upload their malicious code without being noticed by the governance during the voting process.

A potential fix for this is to ensure the `market` parameters are markets instantiated by the factory contract. This can be achieved by checking the address against the `KNOWN_MARKETS` state. An example is demonstrated below:

```
let is_market_known = KNOWN_MARKETS.load(deps.storage, market)?;
```

```
        if !is_market_known {
            return Err(StdError::GenericErr { msg: "incorrect
market".to_string() })
        }
```

This fix should be implemented in the `InitializeLiquidity`, `AddLiquidity`, `RemoveLiquidity`, `Claim`, `ResolveMarket`, and `PlaceOrder` messages to prevent users from maliciously draining the fee funds.

Code Criteria

This section provides an evaluation of specific criteria aspects as described below:

- **Documentation:** Evaluating the presence and comprehensiveness of publicly available or provided explanatory information, diagram flowcharts, comments, and supporting documents to enhance code understanding.
- **Coverage:** Evaluating whether the code adequately addresses all necessary cases and scenarios, ensuring that the intended functionality or requirements are sufficiently covered.
- **Readability:** Assessing how easily the code can be understood and maintained, considering factors such as code structure, naming conventions, and overall organisation.
- **Complexity:** Evaluating the complexity of the code, including factors such as, number of lines, conditional statements, and nested structures.

The status of each criteria is categorised as either **SUFFICIENT** or **NOT-SUFFICIENT** based on the audit assessment. This categorisation provides insights to identify areas that may require further attention and improvement.

Criteria	Status	Notes
Documentation	**SUFFICIENT**	N/A
Coverage	**SUFFICIENT**	N/A
Readability	**NOT-SUFFICIENT**	A few issues are found that affect code readability: 1) The names of used variables should not start with underscores (e.g., `contracts/factory/contract.rs:37`), as underscores are reserved for unused variables. 2) Unused functions in `contracts/market/contract.rs:94-108` should be removed. 3) Run "`cargo clippy`" to increase code readability and maintainability.
Complexity	**SUFFICIENT**	N/A

Findings Summary

Summary Title	Risk Impact	Status
Shares granted to the fee share address are not included in global shares, causing user funds to be stuck	SEVERE	RESOLVED
Users can front-run resolve orders to avoid losing funds, allowing them to have an unfair advantage	SEVERE	RESOLVED
Potential rounding issue causes loss of funds when adding and removing liquidity	MODERATE	RESOLVED
Admins cannot be removed	MODERATE	RESOLVED
Incorrect share variant increased	MODERATE	RESOLVED
Addresses are not validated	LOW	RESOLVED
Incorrect error messages	INFO	RESOLVED
Unused `estimated_profit` variable	INFO	RESOLVED

Findings Technical Details

1. Shares granted to the fee share address are not included in global shares, causing user funds to be stuck

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

In `contracts/market/execute.rs:509-537`, the `place_order` function increases the global shares (`Information.yes_shares`) by the computed shares amount (`quote.amount_out`). This is required as the global shares should include all share amounts correctly.

However, the shares that are granted to the fee address in line 537 (`quote.fees/2`) are not added to the global shares in line 531. This is problematic because the shares cannot be sold due to an underflow error in line 570, preventing users from placing sell orders to withdraw their funds.

This issue also affects other order types:

- Sell "*yes*" orders: `contracts/market/execute.rs:593`
- Buy "*no*" orders: `contracts/market/execute.rs:663`
- Sell "*no*" orders: `contracts/market/execute.rs:727`

Recommendation

Consider updating the global shares to include the shares granted to the fee address. An example fix for buying "*yes*" orders is shown below:

```
info.yes_shares += quote.amount_out.clone() + quote.fees /
Uint128::from(2u128);
```

2. Users can front-run resolve orders to avoid losing funds, allowing them to have an unfair advantage

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

The factory contract admins will call the `ResolveMarket` message when the market has reached its deadline. The admins will specify the market variant's result to be a "*yes*" or "*no*". Depending on the outcome, users who bought shares for the correct variant will receive rewards and lose funds for the other variant.

However, users can front-run the transaction to place sell orders before the admin resolves the market. This enables them to avoid losing funds in markets and gives them unfair advantages over other users.

Recommendation

Consider implementing a `resolve_duration` field in the market contract and implement the logic according to the following:

1. Allow users to `place_order` before `Information.market_end` timestamp has been reached.
2. Only allow admins to `resolve_market` within `Information.market_end` + `resolve_duration` duration. This ensures that users cannot modify their existing orders before the admin resolves the market.
3. Allow users to `place_order` after the `Information.market_end` + `resolve_duration` timestamp has elapsed, regardless of whether the admin has resolved the market. This increases the transparency of the protocol, as admins cannot prevent users from modifying orders if they have not resolved the market.

3. Potential rounding issue causes loss of funds when adding and removing liquidity

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

In `contracts/market/execute.rs:151-154`, the `add_liquidity` function computes the number of shares to mint based on the USDC amount supplied.

The issue is that if the supplied `amount` is small, the computed `shares_to_give` variable may round to zero due to integer truncation. This means the user will not receive any shares while their USDC is taken, causing a loss-of-funds scenario.

This issue also affects the `remove_liquidity` function in `contracts/market/execute.rs:216`, where the USDC amount computed (`amount_to_remove` variable) is zero, but the user's shares amount is decreased incorrectly.

Recommendation

Consider returning an error if the `shares_to_give` or `amount_to_remove` variables are zero.

4. Admins cannot be removed

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

In `contracts/factory/execute.rs:97`, the `add_admin` function allows existing admins to add new admins to the `ADMINS_MAP` state. However, there is no functionality to remove existing admins from the codebase.

As a result, if any of the existing admins are compromised, other admins cannot remove it to reduce the attack surface.

Recommendation

Consider implementing a privileged entry point to remove admins from the `ADMINS_MAP` state.

5. Incorrect share variant increased

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

In `contracts/market/execute.rs:669`, the fee recipient's `fee_shares.no_shares` should be increased by the share amounts because the order type is a buy "*no*" order. The current logic incorrectly increases the share amounts to `fee_shares.yes_shares`, preventing the fee recipient from selling their shares.

Recommendation

Consider replacing `fee_shares.yes_shares` with `fee_shares.no_shares`.

6. Addresses are not validated

RISK IMPACT: LOW	STATUS: RESOLVED

Description

In a few instances of the codebase, the address fields are passed as `Addr`:

- USDC contract address: `contracts/factory/contract.rs:24`
- Fee receiver address: `contracts/factory/contract.rs:25`
- Admin address: `contracts/factory/execute.rs:101`

This is problematic because these addresses are not checked to be valid with the `deps.api.addr_validate` function. Invalid addresses may occur when constructed with the `Addr::unchecked` function.

Recommendation

Consider validating the addresses with the `deps.api.addr_validate` function.

7. Incorrect error messages

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

In a few instances of the codebase, the error messages are raised as "*Price impact must not be more than 3%*":

- `contracts/market/execute.rs:575`
- `contracts/market/execute.rs:644`
- `contracts/market/execute.rs:707`

This is incorrect because the validation checks that the max impact does not exceed 5%, not 3%.

Recommendation

Consider updating the error messages to "*Price impact must not be more than 5%*".

8. Unused `estimated_profit` variable

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

In `contracts/market/execute.rs:786`, the `quote` function implements an `estimated_profit` variable, but it is not used throughout the execution and always returns as zero value in line `890`.

Recommendation

Consider implementing the necessary logic to estimate profits or removing it from the codebase.

Document Control

Version	Date	Notes
-	18th February 2025	Security audit commencement date.
0.1	28th February 2025	Initial report with identified findings delivered.
0.5	1st March 2025	Fixes remediations implemented and reviewed.
1.0	2nd March 2025	Audit completed, final report delivered.

Appendices

A. Appendix – Risk assessment methodology

SCV-Security employs a risk assessment methodology to evaluate vulnerabilities and identified issues. This approach involves the analysis of both the LIKELIHOOD of a security incident occurring and the potential IMPACT if such an incident were to happen. For each vulnerability, SCV-Security calculates a risk level on a scale of 5 to 1, where 5 denotes the highest likelihood or impact. Consequently, an overall risk level is derived from combining these two factors, resulting in a value from 10 to 1, with 10 signifying the most elevated level of security risk

Risk Level	Range
CRITICAL	10
SEVERE	From 9 to 8
MODERATE	From 7 to 6
LOW	From 5 to 4
INFORMATIONAL	From 3 to 1

LIKELIHOOD and **IMPACT** would be individually assessed based on the below:

Rate	LIKELIHOOD	IMPACT
5	**Extremely Likely**	Could result in severe and irreparable consequences.
4	**Likely**	May lead to substantial impact or loss.
3	**Possible**	Could cause partial impact or loss on a wide scale.
2	**Unlikely**	Might cause temporary disruptions or losses.
1	**Rare**	Could have minimal or negligible impact.

B. Appendix – Report Disclaimer

This report should not be regarded as an "endorsement" or "disapproval" of any specific project or team. These reports do not indicate the economics or value of any "product" or "asset" created by a team or project that engages SCV-Security for a security review. The audit report does not make any statements or warranties about the code's utility, safety, suitability of the business model, regulatory compliance of the business model, or any other claims regarding the fitness of the implementation for its purpose or its bug-free status. The audit documentation is intended for discussion purposes only. The content of this audit report is provided "as is," without representations and warranties of any kind, and SCV-Security disclaims any liability for damages arising from or in connection with this audit report. Copyright of this report remains with SCV-Security.

THANK YOU FOR CHOOSING

SCV SECURITY

scv.services

contact@scv.services