AUDIT REPORT

xNinja Lab

cw-controller

Prepared by SCV-Security

On 2nd March 2024

Table of Contents

Introduction

SCV has been engaged by xNinja Lab 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 xNinja cw-controller contract facilitates controller functionality for xNinja.Tech. It allows users to borrow XNJ governance tokens against INJ token collateral which can then be converted into ELEM tokens for in-game use. Users can use ELEM tokens in-game to buy chests, items, or to level up their Ninja and can claim additional ELEM tokens based on gameplay rewards. The in-game ELEM tokens can then be converted back to XNJ tokens and returned to receive back any INJ collateral that's available to the user. The xNinja cw-controller contract interfaces with the end user through a backend app that creates the execution messages for borrowing XNJ, repaying XNJ, and claiming ELEM.

Submitted Codebase

cw-controller	
Repository	https://github.com/xninja-lab/rust-csld-contracts
Commit	`32641fa6a8bf3d3d7b3c663c125e6bf6aad0c245`
Branch	`main`
Contract	`cw-controller`

Revisions Codebase

cw-controller	
Repository	https://github.com/xninja-lab/rust-csld-contracts
Commit	`c55944d238808d0e43c6f7b951c79546d2d08e97`
Branch	`main`
Contract	`cw-controller`

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 xNinja Lab. 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.

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**	The protocol did not provide a detailed technical specification, but did include high level documentation that covered the general application flow.
Coverage	**NOT-SUFFICIENT**	Test coverage is considered not sufficient due to core contract functionality not having any corresponding tests. The current coverage only extends to 61.17% of the code. Covered further in the Audit Observations section.
Readability	**SUFFICIENT**	The codebase had good readability overall and utilised many Rust and CosmWasm best practices.
Complexity	**SUFFICIENT**	The contract complexity was managed well. However the out of scope components such as the backend application were not considered in this measurement.

Findings Summary

Summary Title	Risk Impact	Status
`execute_borrow` relies on backend application for price feed	CRITICAL	ACKNOWLEDGED
`execute_stop_borrow` allows the admin to invalidate all borrowing positions	SEVERE	ACKNOWLEDGED
`Elem` to `XNJ` conversions will be floored due to uint division	SEVERE	PARTIALLY RESOLVED
`execute_update_config_params` can introduce state inconsistencies	SEVERE	ACKNOWLEDGED
Admin can send contract funds to any address	MODERATE	ACKNOWLEDGED
Centralization risk for key compromise	MODERATE	ACKNOWLEDGED
If admin is unset it cannot be reset	MODERATE	RESOLVED
`repaying_period` is never enforced	LOW	RESOLVED
Users are not removed from `STAKE_INFO`	LOW	ACKNOWLEDGED
`Instantiate` function emits default attributes	INFO	RESOLVED

Audit Observations

The audit observations section is intended to present potential findings that are related to the underlying design of the protocol and would require underlying design changes to remediate that may change the overall functioning of the protocol. SCV asks that the client formulate responses to add context to validate or invalidate the following concerns.

1. Minimal testing for core contract functionality

Some of the core contract functionality is untested. It would be beneficial to add integration tests verifying happy path functionality for the following execution messages and also a round trip integration test following user funds from the borrowing to repayment stage.

- Individual function happy path integration tests
  - `execute_borrow`
  - `execute_repay`
  - `execute_claim_elem`
  - `execute_claim`
- Round trip integration test asserting expected params and balances along the way
  - `execute_borrow => execute_convert_xnj_to_elem => execute_claim_elem => execute_convert_elem_to_xnj => execute_repay`

2. Hooks should be removed from contract

Since the hooks are no longer intended to be used to store membership information in other contracts, they should be removed from the cw_controller contract.

Findings Technical Details

1. `execute_borrow` relies on backend application for price feed

RISK IMPACT: CRITICAL	STATUS: ACKNOWLEDGED

Revision Notes

The xNinja team has stated that this architecture has been decided upon to strictly control the LTV ratio.

Description

The `execute_borrow` function in `contracts/cw-controller/src/contract.rs:146` accepts the value of `inj_price_usd` as a function parameter from the caller, which in this case is an off-chain backend application. This architecture makes a trust assumption that the backend application is able to provide reliable price data. This trust assumption is problematic because it cannot be guaranteed that the price used is the current price in the block in which the borrow is executed.

For example, the `verify_nonce_and_timestamp` function checks to ensure that the timestamp provided with the message is within 300 seconds of the current blocktime, plus or minus. This means that the `xnj_received` can differ, potentially dramatically, from the true USD value at the block in which the transaction is executed.

There are multiple ways in which attackers can take advantage of this architecture as well as multiple scenarios in which volatile market conditions will result in unfair execution to both the protocol and the users.

For example an attacker could DDOS the backend application and manipulate price feed timing and inclusion similarly to the Levana Price Delta Attack . This could allow them to control the general time in which the borrow is executed. There is a potentially large attack surface, even in a non-attack scenario where markets are highly volatile this architecture could result in worse execution for the protocol.

It is important to note that the backend application is out of scope and SCV has not reviewed the backend components associated with price feeds or borrowing.

Recommendation

We recommend moving the price feed logic on-chain by implementing an oracle feed for the INJ USD value. This will ensure that the borrowing price matches the price at the block of execution. Additionally, we recommend reducing the borrowing timestamp validation period to a smaller value.

2. `execute_stop_borrow` allows the admin to invalidate all borrowing positions

RISK IMPACT: SEVERE	STATUS: ACKNOWLEDGED

Description

The `execute_stop_borrow` function in `contracts/cw-controller/src/contract.rs:236` allows the admin to invalidate all borrowing positions. With this function call, users will no longer be able to redeem their staked `INJ`. If a malicious or compromised admin were to call this function, it would effectively erase all borrowing positions in the controller contract. Additionally, there is a potential for inconsistent borrowing state changes as the `borrowing_period` flag can also be modified during a config update.

Recommendation

We recommend modifying the scope of the `execute_stop_borrow` function to only change the `borrowing_period` rather than removing all user borrowing positions.

3. `Elem` to XNJ conversions will be floored due to uint division

RISK IMPACT: SEVERE	STATUS: PARTIALLY RESOLVED

Revision Notes

The xNinja team has updated the elem to xnj conversion to be 1 to 1 and will enforce a validation to ensure the conversion is not 0 on the backend. Additionally this message now implements signature validation which ensures the signed messages are only coming from the backend application.

Description

The `execute_convert_elem_to_xnj` function in `contracts/cw-controller/src/contract.rs:456` calculates the claimed `xnj_amount` by performing a multiplication of the ratio of `elem_to_xnj_rate` with a denominator of `10^18`. This is problematic because the multiplication may result in `0` under certain cases where the `elem_amount` is not sufficiently large.

Using values based off of other test cases it was determined that based on the assumed conversion rate, conversions with an amount less than `1_000_000` elem tokens will floor `0` and result in a `xnj_amount` of `0`. This is especially problematic because the function will not error. Instead a CLAIM will be created with an amount of 0 without any error.

Recommendation

We recommend throwing an error if the `xnj_amount` amount is determined to be `0`. This will effectively enforce a minimum claim amount. The following test case can be used as reference:

- https://gist.github.com/scvsecurity/96a7214365538563482aaed0054d4dd9

4. `execute_update_config_params` can introduce state inconsistencies

RISK IMPACT: SEVERE	STATUS: ACKNOWLEDGED

Description

The `execute_update_config_params` function in `contracts/cw-controller/src/contract.rs:787` allows the admin to update any of the config parameters. This is problematic because a malicious or compromised admin updating these values when there is an active state in the contract would introduce inconsistencies.

- Modifying `elem_to_xnj_rate` or `loan_rate` could immediately dilute user token value
- If `tokens_per_weight` is updated, then it will dilute all existing weights in `MEMBERS`. Additionally, if `tokens_per_weight` is set to `0`, then any subsequent call to `execute_borrow` or `execute_repay` would result in a divide by zero error when calling `calc_weight`
- Updating `denom_xnj` after users have borrowed could potentially lock their funds in a borrow position that they cannot repay to redeem their original collateral
- `borrowing_period` should only be updated in its own execute message to emit proper attributes
- Updating `cfg.denom` would mean that users will receive a different coin than they sent when repaying a borrowing position. Also, the contract will not have enough `denom` at a point to issue repayments because its balance would be 0.

Recommendation

We recommend adding a validation to the `execute_update_config_params` function that ensures that the protocol is not active before config changes are being made. This would ensure that parameter updates could not interfere with

active borrowing positions. Additionally, it is recommended to implement protocol governance to manage critical actions such as config updates.

5. Admin can send contract funds to any address

RISK IMPACT: MODERATE	STATUS: ACKNOWLEDGED

Description

The `execute_add_to_treasury` function in `contracts/cw-controller/src/contract.rs:726` allows the admin to send a specified amount of `cfg.denom` to a caller specified address. This is a dangerous pattern because the caller may specify any target. In the event of a malicious or compromised admin, the contract balance of `cfg.denom` can be sent to any address. It is best practice to store the treasury address as a config variable of the contract to ensure that funds are only sent to whitelisted addresses.

Additionally, the target is not validated to be a valid cosmos address, this is an informational point because the bank module will reject the transaction if the address is invalid, but it is best practice to provide an error at the contract level for this case.

Recommendation

We recommend storing the treasury address as a config variable of the contract to ensure that funds are only sent to whitelisted addresses. This will also resolve the unvalidated target issue.

6. Centralization risk for key compromise

RISK IMPACT: MODERATE	STATUS: ACKNOWLEDGED

Description

The functionality of the `cw-controller` contract is heavily dependent on the parameter `config.public_key` which is an updatable parameter. The Admin can call `execute_update_config_params` to update the `public_key` to an unvalidated binary value. While the backend program is out of the scope of this audit, all efforts should be made to limit the impact of a key compromise even though this is a very low likelihood event.

This could be potentially catastrophic in the event of a misconfiguration or an admin compromise. After `execute_update_config_params` has been executed, calls can be made immediately with the new signature. If protocol architecture requires the centralized components, it is best practice to implement time lock delay functionality as well as having protocol governance control these critical parameters.

Recommendation

We recommend implementing steps to limit the likelihood and impact of a compromised signing key. This includes implementing protocol governance to control the configuration and update of key parameters. Additionally a two-step transfer process can be implemented to ensure that the signing key is being transferred to the correct address and to control this update through protocol governance. Finally, the protocol can implement timelock functionality to enforce time delays for critical actions after a singing key or admin transfer occurs.

7. If admin is unset it cannot be reset

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `execute_update_admin` function in `packages/controllers/src/admin.rs:68` allows the admin address to be set to `None`. If the admin is set to `None`, it will effectively disable important functionality such as burning tokens, stopping borrowing, adding to the treasury, modifying hooks and calling `UpdateMod`.

It is also important to note that this is also the case if admin is set to `None` during the instantiation.

Recommendation

We recommend ensuring that the admin value is not set to `None` before it is passed to the CW-Controller admin package.

8. `repaying_period` is never enforced

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `repaying_period` config parameter is defined in `contracts/cw-controller/src/state.rs:21` but is never used by the controller contract during the repayment process.

Recommendation

We recommend implementing functionality for `repaying_period` to give more stability and better controls on open borrow positions or if this feature is not intended to be used it should be removed.

9. Users are not removed from STAKE_INFO

RISK IMPACT: LOW	STATUS: ACKNOWLEDGED

Description

The `execute_repay` function in `contracts/cw-controller/src/contract.rs:269` allows users to repay their loans and receive back their INJ collateral. In the situation where a user makes a complete repayment their entire collateral is returned and the update_membership function will remove the user from `MEMBERS` in line `contracts/cw-controller/src/contract.rs:673`. Even if the user has completed a full repayment and has zero amounts of `total_xnj_received` and `total_inj_staked`, the entry will not be removed from `STAKE_INFO`. It is best practice to maintain a clean state and prune the entries that are no longer needed.

Recommendation

We recommend removing users from `STAKE_INFO` if they no longer have a borrowing position.

10. `Instantiate` function emits default attributes

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `instantiate` function in `contracts/cw-controller/src/contract.rs:57` emits empty default attributes. It is best practice to emit detailed attributes whenever a state change occurs.

Recommendation

We recommend emitting detailed attributes in the `instantiate` function.

Document Control

Version	Date	Notes
-	19th February 2024	Security audit commencement date.
0.1	24th February 2024	Initial report with identified findings delivered.
0.5	1st March 2024	Fixes remediations implemented and reviewed.
1.0	2nd March 2024	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