AUDIT REPORT

Ninja Blaze Jackpot Contract

Prepared by SCV-Security

On 20th December 2023

Table of Contents

Introduction

SCV has been engaged by Ninja Blaze 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

Ninja Jackpot contract is a jackpot game contract created with the Slyvia framework that allows users to deposit funds to bet themselves as the winner and receive all the prizes. It relies on Nois and Drand for randomness generation to pick the winner among all bidders.

Submitted Codebase

contracts	
Repository	https://github.com/NinjaBlazeApp/contracts
Commit	`b1597cedd856ec6aca4f9e19065ca3f7489403eb`
Branch	`main`

Revisions Codebase

contracts	
Repository	https://github.com/NinjaBlazeApp/contracts
Commit	`b2332509d918e42c9311ce3ec8ce15c6dda4ea1a`
Branch	`main`

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 Ninja Blaze. 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**	N/A
Coverage	**SUFFICIENT**	Testing coverage is considered sufficient with 82.32% coverage 135/164 lines covered.
Readability	**SUFFICIENT**	The codebase had good readability overall.
Complexity	**SUFFICIENT**	N/A

Findings Summary

Summary Title	Risk Impact	Status
Replaying the `start_draw` message causes the contract to lose `randomness_provider_fee` funds	MODERATE	ACKNOWLEDGED
The randomness requested does not include a safety margin	MODERATE	RESOLVED
The game cannot be concluded if there are too many bidders	MODERATE	RESOLVED
Incorrect migration validation and event emitted	MODERATE	RESOLVED
The fee recipient address is not validated	LOW	RESOLVED
Updating the bid denom causes state inconsistency	LOW	RESOLVED
Fee is not checked to be between 1% and 99%	LOW	RESOLVED
Existing bidders from the previous game remain in storage	INFO	ACKNOWLEDGED

Findings Technical Details

1. Replaying the `start_draw` message causes the contract to lose `randomness_provider_fee` funds

RISK IMPACT: MODERATE	STATUS: ACKNOWLEDGED

Revision Notes

The team has mentioned that they plan to have a worker call the `start_draw` method of the smart jackpot contract and send funds to pay Nois. They will also implement a mechanism for Workers to prevent too much money from being spent. The current implementation allows them to repeat the request to Nois if the first one fails.

Description

The `start_draw` function in `src/contract.rs:168` sends `config.randomness_provider_fee` to the randomness provider when requesting randomness as part of the fee payment. After the randomness is requested, there is a period before the `NoisCallback` is called in line `177`.

During the period, anyone can replay the `start_draw` function to request randomness from the Nois proxy contract. Even though the callback will eventually fail in lines `192` to `196`, the `config.randomness_provider_fee` is paid and not refunded, causing a loss of funds scenario.

Recommendation

Consider preventing the `start_draw` function from being called when `game.state` is `GameState::Drawing`.

2. The randomness requested does not include a safety margin

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `start_draw` function in `src/contract.rs:160-163` requests randomness from the Nois proxy contract with the published randomness time after the `game.draw_timestamp`. This is problematic because the requested timestamp fails to take into account that the BFT block time can be behind the timestamp, potentially causing the published randomness to be lesser than the `game.draw_timestamp` and ultimately erroring in line `204`, preventing a game from concluding correctly.

Recommendation

Consider applying a safety margin of 5 seconds to ensure the requested timestamp is published after the `game.draw_timestamp`. An example of the implementation can be found here.

3. The game cannot be concluded if there are too many bidders

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `nois_receive` function in `src/contract.rs:212-220` retrieves all bidders, iterates over it, and calls the `select_from_weighted` function with the randomness seed to pick the winner. The problem occurs when too many bidders join a game, causing the iteration to fail due to an out-of-gas error and preventing the game from being concluded.

Recommendation

Consider implementing a maximum limit of bidders that can join for a game to prevent out-of-gas errors.

4. Incorrect migration validation and event emitted

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `migrate` function in `src/contract.rs:268` only allows contract migration from the same contract name. However, as the `set_contract_version` is called first in line 269, the validation in lines 272 to 276 will always become true because the contract version is already mutated in line 269, allowing other contracts with different contract names to be migrated into the Jackpot contract.

Furthermore, the `old_contract_version` attribute emitted in line 285 will be the same as `CONTRACT_VERSION`, which is incorrect.

Recommendation

Consider moving line 269 below line 276 so the validation and event emit work correctly.

5. The fee recipient address is not validated

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `update_config` function in `src/contract.rs:256-260` validates the manager and randomness provider address when updating the configuration.

However, the `config.fee_recipient` address is not validated, potentially causing the `fee_msg` in line `98` to fail to be dispatched properly due to an incorrect address format.

Recommendation

Consider validating the fee recipient address in the `update_config` function.

6. Updating the bid denom causes state inconsistency

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `update_config` function in `src/contract.rs:250` allows updating the `config.denom`, which is the denom bidders send to join a game in line 88. If the manager updates the denom when existing bids are in `game.win_bank`, the final amount will be incorrect because two different denoms' amounts are added together. Furthermore, the contract will show that the `game.win_bank` state owns the new denom amount, but the contract balance holds the old denom instead.

Recommendation

Consider removing the option for the manager to update `config.denom` in the contract.

7. Fee is not checked to be between 1% and 99%

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `update_config` function is called to update the `config.fee` value in `src/contract.rs:250`. The problem occurs when there is no check that the fee is not validated to be 1% to 99%.

If the fee percentage is configured as 0%, the `join` function in line `96` will compute the `fee` as zero. This will cause the `fee_msg` to fail to be dispatched because Cosmos SDK does not allow the transacting of zero native token amounts, preventing bidders from joining the jackpot.

If the fee percentage is configured as 100%, the `game.win_bank` in line `108` will increase by zero for all bidders. This will cause the `win_bankmsg` to fail in line `222` when concluding a jackpot because it would send zero native token amount to the winner, essentially DoS'ing the contract because new games cannot be started.

If the fee percentage is configured over 100%, an overflow will occur in line `108`, preventing bidders from joining the jackpot.

Recommendation

Consider updating the `update_config` function to ensure the `config.fee` is between 1% and 99% percentage.

8. Existing bidders from the previous game remain in storage

RISK IMPACT: INFORMATIONAL	STATUS: ACKNOWLEDGED

Revision Notes

The team has mentioned that they implemented soft cleaning to reduce gas consumption. Hard cleaning reduces the number of bets per round.

Description

The `clear` function in `src/storage.rs:67` overwrites the count to zero without removing the items state that contains all the bidders' information. This works well with the actual contract functionality, as only the old bidders are ignored because the `get_all` function only fetches until the latest count variable in lines 31 and 32.

The issue occurs when the first game has more bidders than the second game. In this case, the `self.items` state will hold more bidders (from the first game) than the `count` state (handled by the second game). If a user or third-party contract performs a raw query to the contract's storage to fetch all the bets, it will be misleading to them in a way that the second game bidder includes old bidders from the first game, which is incorrect.

Please see this Gist link for a proof of concept test case.

Recommendation

Consider calling `self.items.clear(storage)` in the `clear` function to remove the previous game's bidders. Additionally, consider implementing a smart query that returns all bidders for the current game.

Document Control

Version	Date	Notes
-	4th December 2023	Security audit commencement date.
0.1	11th December 2023	Initial report with identified findings delivered.
0.5	19th December 2023	Fixes remediations implemented and reviewed.
1.0	20th December 2023	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