Audit Report

Ninja Blaze: Double

v1.0

May 6, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Ninja Blaze to perform a security audit of Double.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/NinjaBlazeApp/contracts
Commit	`f87bc838c73aa80bf7f7911375a8229062edb7fb`
Scope	The `contracts/double` contract and any imports from `packages` were in scope.
Fixes verified at commit	`42abfaae3bf421b13104efe52b1a78663cbf250b` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Double is a fast-paced on-chain game in which players can place bets on three colors: green, red, or black. Additionally, if the jackpot is won, the amount is distributed to all participants of the current round in proportion to their bets, regardless whether a bet was won or lost.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	-
Test coverage	N/A	We were not able to determine the test coverage due to compilation errors while using `cargo tarpaulin`.

Summary of Findings

No	Description	Severity	Status
1	Invalid user-provided addresses can lock the contract	Critical	Resolved
2	A game can be drawn multiple times in the same round, resulting in a jackpot reset and locked funds in the contract	Critical	Resolved
3	Additional fees are not properly accounted for in the bet	Major	Acknowledged
4	Potential risk of insufficient funds for user reward payouts	Major	Acknowledged
5	The configuration's denomination should not be modifiable	Minor	Resolved
6	Lack of configuration validation	Minor	Partially Resolved
7	Overflow checks not enabled for release profile	Minor	Resolved
8	Game state is not explicitly checked	Minor	Resolved
9	Configuration update pattern is error-prone	Informational	Acknowledged
10	Unused code	Informational	Resolved
11	Incorrect or outdated documentation	Informational	Resolved
12	"Migrate only if newer" pattern is not followed	Informational	Acknowledged
13	Empty reply entry-point	Informational	Acknowledged
14	Use of magic numbers decreases maintainability	Informational	Acknowledged
15	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
16	Lack of role-based access controls for the pausing mechanism	Informational	Acknowledged

Detailed Findings

1. Invalid user-provided addresses can lock the contract

Severity: Critical

The `join` function in `contracts/double/src/contract.rs:89-188` takes an optional user-provided `player` argument as the beneficiary of the bet being placed. However, this address is not validated before being saved.

The addresses saved in the `bets` storage vector are used for reward and jackpot distribution in the `processing` function. An invalid address included in the `BankrollExecMsg::Payout` and `BankMsg::Send` messages, crafted in lines `306-309` and `326-329`, will result in an error that will revert the transaction. Therefore, this makes the `processing` function unusable and prevents any fund distribution.

Recommendation

We recommend validating the address before saving it to storage.

Status: Resolved

2. A game can be drawn multiple times in the same round, resulting in a jackpot reset and locked funds in the contract

Severity: Critical

In `contracts/double/src/contract.rs:191-254`, the `draw` function determines the winning color via the provided randomness beacon, consisting of the `round` and the DRAND `signature`.

However, the `draw` function can be called repeatedly for the same game round. While the winning color is deterministically determined and remains the same, the jackpot is reset, forfeiting any previously accumulated jackpot funds.

Recommendation

We recommend ensuring that the game's state differs from `GameState::Distributing` when drawing to ensure that the current game has not already been drawn.

Status: Resolved

3. Additional fees are not properly accounted for in the bet

Severity: Major

When joining the current game via the `join` function in `contracts/double/src/contract.rs:87-192`, the tokens sent are added to the `bet.amount`, and then to the pool of funds that will participate in it, called `game.pool`. The pool is increased directly by the amount of funds transferred by the user. Later, in line `160`, the funds are reduced by the `fee`, calculated based on the percentage set in the configuration, as well as by `next_jackpot_pool`, which is 0.05% of the deposit value, which will serve as the jackpot in subsequent games.

The above assumptions are problematic because the amount of funds stored as `bet.amount` and `game.pool` will not be equal to the actual amount of funds allocated to pay user rewards. This may result in discrepancies between the actual amount of funds and what the protocol believes it has, leading to payout problems for recent winners when the bankroll contract does not have enough tokens.

Recommendation

We recommend changing `bet.amount` and in consequence also `game.pool` to equal the number of tokens sent minus the `fee` and `next_jackpot_pool` value.

Status: Acknowledged

4. Potential risk of insufficient funds for user reward payouts

Severity: Major

The contract logic defines rewards as twice or fourteen times the user's bet, depending on their chosen color. When there are more winners than losers in a game round, there is the likelihood that user rewards will significantly exceed the available pool from a given game, which will have to be covered by the protocol itself by having the Ninja Blaze team manually replenish the `bankroll` contract.

Otherwise, if the `bankroll` contract holding the deposits does not have enough funds, the distribution will fail.

Recommendation

We recommend implementing a mechanism to verify whether the contract will be able to pay out the total amount of funds. If the funds are not sufficient, we recommend paying out proportional rewards.

Status: Acknowledged

5. The configuration's denomination should not be modifiable

Severity: Minor

The `update_config` function in `contracts/double/src/contract.rs:406-421` does not restrict which fields can be updated. Therefore, the `denom` field can be freely changed, both during a game and between games.

Modifying this field directly impacts user funds, as reward distribution within a game and any accumulated jackpot will not match the deposits. In addition, trying to distribute a jackpot amount of a new denomination can result in an out-of-funds error, which will lock both the reward and jackpot funds.

We classify this issue as minor since only the contract owner, a trusted entity, can introduce it.

Recommendation

We recommend making the `denom` field non-updatable.

Status: Resolved

6. Lack of configuration validation

Severity: Minor

The `update_config` function in `contracts/double/src/contract.rs:406-421` does not perform sufficient validation of most configuration fields:

- If the `fee` has a zero value, it will fail in line `172`. In addition, if its maximum value is larger than 1 minus the jackpot's fee, an underflow error will occur in line `160`.
- If the `fee_recipient` is not validated, the fee collection message in line `166` will fail, making the `join` function unusable.
- The `denom` should be validated by querying the token factory contract and ensuring its supply exceeds zero. An incorrect denomination will make the contract unusable.
- The `min_amount` should be greater than zero to avoid free participation.
- If the `max_bet` limits are below the `min_amount` the contract will be unusable. It is recommended that they are at least several times larger to allow several users to participate.
- If `game_time` is zero, the contract will be unusable. It is recommended to set a suitable minimum. For example, the documentation mentions that games are expected to last around 30 seconds.
- If the `jackpot_chance` is zero, the jackpot will hit every game.
- If `draw_safe_margin` is equal to or greater than `game_time`, joining the game will be impossible, making it unusable.
- If `batch_size` is too large, the for loop in line `282` could cause an out-of-gas exception, preventing distribution. Aside from that, it is unnecessary to declare it as a signed integer.

Recommendation

We recommend validating the configuration values following the comments above.

Status: Partially Resolved

7. Overflow checks not enabled for release profile

Severity: Minor

The `contracts/double/Cargo.toml` package does not enable `overflow-checks` for the release profile.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

8. Game state is not explicitly checked

Severity: Minor

The `processing` function in `contracts/double/src/contract.rs:257` should fail if the game is not in the `Distributed` state. While this condition is not explicitly checked, line `266` unwraps a value that is saved alongside the state update performed in the `draw` function, resulting in an error if not set.

While we did not find an exploitable security concern with the lack of game state validation, not explicitly checking expected states can be error-prone. Also, without an explicit check, users are not provided with a meaningful error message.

In addition, the `Drawing` state defined in `contracts/double/src/state.rs:114` is not used through the codebase.

Recommendation

We recommend explicitly checking that each function can only be called if the game is in the expected state. In addition, unused states should be removed for clarity.

Status: Resolved

9. Configuration update pattern is error-prone

Severity: Informational

The `admin_update_config` function in `contracts/double/src/contract.rs:70-86` takes a completely new `ContractConfig` struct among its input arguments rather than allowing partial updates of selected parameters. Requiring the operator to submit a whole new struct to modify a single field decreases user experience and is error-prone.

Recommendation

We recommend allowing the sender to specify which parameter to update as an `Option` instead of accepting a new `ContractConfig` struct.

Status: Acknowledged

10. Unused code

Severity: Informational

The audited scope contains unused code, which may indicate that the logic has changed or that assumptions do not match the current implementation. These are:

- `UnauthorizedReceive` error in `contracts/double/src/error.rs:37-38`
- `Drawing` status defined as part of `GameState` enum in `contracts/double/src/state.rs:114`

Recommendation

We recommend adapting the logic to implement and use the above elements or remove them from the codebase.

Status: Resolved

11. Incorrect or outdated documentation

Severity: Informational

The implementation within the scope of this audit deviates from the documentation shared by the development team. While we did not identify a security issue connected to these deviations, keeping up-to-date documentation is important for users to trust the protocol. The following pieces should be corrected:

- The documentation states: “the color that is last in the game will win”. However, the winning color is randomly derived from an oracle.

- The contract identifies the colors as `Red`, `Black`, and `Green`, as opposed to the documentation, which uses `Purple`, `Blue`, and `Green`.
- The documentation describes `Blue` as "the most frequent winner". However, this differs from the implementation, where both non-green colors have the same probability of winning (7/15).

Recommendation

We recommend updating the documentation so it matches the contract's implementation.

Status: Resolved

12. "Migrate only if newer" pattern is not followed

Severity: Informational

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure the migration is only performed if the supplied version is newer.

Recommendation

We recommend following the "migrate only if newer" pattern defined in the CosmWasm documentation.

Status: Acknowledged

13. Empty reply entrypoint

Severity: Informational

The `reply` function in `contracts/double/src/contract.rs:447-449` returns a successful response without any other logic being executed. Although we did not identify a security issue, it is best security practice to avoid empty handlers as they are error-prone.

Recommendation

We recommend explicitly matching the reply code and reply result that should be handled, emitting relevant event attributes when handled properly, and erroring otherwise.

Status: Acknowledged

14. Use of magic numbers decreases maintainability

Severity: Informational

In `contracts/double/src/contract.rs:131`, `216`, `223`, `291`, and `293`, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Acknowledged

15. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

16. Lack of role-based access controls for the pausing mechanism

Severity: Informational

The codebase implements a pausing mechanism, which is in line with best practices. However, all of the contract's administrative functions are centralized in the admin role, which goes against the principle of least privilege.

Segregating the pauser role has the additional benefit of swifter reactions in case of need when assigned to an EOA compared to the admin that might be managed by a multisig or a governance contract.

Recommendation

We recommend implementing a separate pauser role that can turn the pausing mechanism on and off.

Status: Acknowledged