BURNICORN

SMART CONTRACT AUDIT REPORT

IMMUNEBYTES

02 May 2023

Introduction 3

About Burnicorn 3

About ImmuneBytes 3

Documentation Details 3

Audit Goals 4

Audit Process & Methodology 5

Audit Details 6

Security Level References 7

Finding 8

Critical Severity Issues 9

High Severity Issues 9

Medium severity issues 9

Low severity issues 11

Informational 14

Automated Test Results 15

Concluding Remarks 18

Disclaimer 18

Introduction

About Burnicorn

Burnincorn is a Web3 arcade game, in which players will pay a small fee in LUNC to play. The majority of that fee is burned, and a portion is sent to a rewards wallet for the daily jackpot. The player will control a robot unicorn and try to collect as many coins as possible before the game ends. At the end of the day, the player with the highest score (most coins collected) will be sent the jackpot. The high scores and jackpot reset daily.

Visit https://game.burnicorn.io/ to know more about it.

About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The team has provided the following doc for audit:

1. https://immunebytes.notion.site/Standard-Contract-Burnicorn-1016134fd4374d9c9c892dbfff3f201c

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include

    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project, starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safely used third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

A team of independent auditors audited the code, including -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

Project Name	Burnicorn
Platform	Terra
Languages	Rust
GitHub Link	https://github.com/adeelJaved-5/Distributor-LUNA-Contract
Commit - Final Audit	fdb0a79b5869dbfd373acdc187dfa7dd8a260729

Security Level References

Every issue in this report was assigned a severity level from the following:

CRITICAL

Issues may result in fund leakage or incorrect fund allocation.

HIGH

Issues affecting the business logic, performance, and functionality.

MEDIUM

Issues could lead to data loss or other manipulations.

LOW

Issues around smart contract code upgradability, libraries, and others.

INFORMATIONAL

Issues which can further improve the code on gas optimizations and reusability.

Issues	Critical	High	Medium	Low	Informational
Open	-	-	-	2	-
Closed	-	-	1	3	1
Acknowledged	-	-	1	-	-

Finding

#	Findings	Risk	Status
1.	Insufficient Essential Queries	Medium	Acknowledged
2.	Insufficient input validation on amount parameter	Medium	Fixed
3.	Insufficient input validations for the execute_deposit coin denomination	Low	Fixed
4.	Inexistent Event Emission in Critical Functions	Low	Pending
5.	Potential Arithmetic Underflow in Withdraw Function	Low	Fixed
6.	Inappropriate Address Validation in receiver Parameter	Low	Pending
7.	Direct access to the funds array	Low	Fixed
8.	Inexistent migration functionality	Informational	Acknowledged

Critical Severity Issues

No issues were found.

High Severity Issues

No issues were found.

Medium severity issues

1. **Vulnerability name: Insufficient Essential Queries**

Description: The contract lacks essential queries that might be useful for some users. The current queries provided in the contract appear to cover the essential information related to the contract's purpose. However, depending on the specific use case, some queries might be missing, such as:

Query for individual account balances: You could add a query that takes an account address as input and returns the account's balance (if applicable). This query would allow users to check their balances without having to consult an external source.
Query for transaction history: You could add a query that returns a list of transactions related to the contract, such as deposits, withdrawals, or fund distribution events. This would provide users with a more comprehensive overview of the contract's activities.
Query for contract version: As the contract has a set_contract_version function, you could add a query that returns the current contract version. This would allow users to verify the version of the contract they are interacting with.
Query for contract status: If there's a possibility that the contract's operations could be paused or halted under certain conditions, you could add a query that returns the current status of the contract (e.g., active, paused, or halted).

Recommendation: Consider adding essential queries to the contract's functionality to provide more insights into the contract's state. These queries include a query for individual account balances, a query for transaction history, a query for contract version, and a query for contract status. This would enhance the contract's functionality and provide users with a more comprehensive overview of the contract's activities.

Status: Acknowledged

Developer's Response:
We are storing data off-chain and have no need for queries and history to be maintained onchain. Moreover, in the scope of development, there is no need for pausable or active status. Thus it can be ignored.

2. **Insufficient input validation on amount parameter**

Description: The contract's function allows the sender to set the amount parameter without performing any validation beyond checking if the sender is authorized to set the amount. This lack of validation on the amount parameter could lead to unexpected behavior if the parameter is not within a certain range or does not follow certain constraints.

Recommendation: To prevent unexpected behavior, it is recommended to add additional validation to the amount parameter. This can be achieved by adding a simple if statement that checks if the amount parameter is within a certain range or follows certain constraints, depending on the specific requirements of the contract.

Status: Amended
The team has fixed the issue.

Low severity issues

1. Insufficient input validations for the execute_deposit coin denomination

Description: In the execute_deposit function, it is assumed that the coin denomination being deposited is "uluna". This assumption is made implicitly when constructing the burn_amt, jack_amt, and dev_amt vectors. The denomination of the deposited coin is taken from info.funds[0].denom, but there is no check to ensure it is "uluna" or any other specific denomination before processing the deposit.

Recommendation: To fix this issue, you should validate the coin denomination before processing the deposit. You can add a check for the expected denomination (e.g., "uluna") and return an error if the deposited coin does not match the expected denomination. Here's an example of how you could modify the execute_deposit function:

```
fn execute_deposit(
   deps: DepsMut,
   info: MessageInfo,
) -> Result<Response, ContractError> {
   let config = CONFIG.load(deps.storage)?;
   let record = RECORD.load(deps.storage)?;

   // Validate the denomination of the deposited coin
   let expected_denom = "uluna";
   if info.funds[0].denom != expected_denom {
      return Err(ContractError::InvalidDenomination {});
   }

   if info.funds[0].amount != config.amount {
      return Err(ContractError::Invalid {});
   }
   // Rest of the code remains unchanged
   // ...
}
```

Status: Amended
The team has fixed the issue.

2. Inexistent Event Emission in Critical Functions

Description: The contract does not emit any events in its functions, which can lead to a lack of transparency and make it difficult for users to track the actions performed by the contract. While the contract adds attributes to the response object using the add_attribute method, these attributes are not emitted as events. For example, in the execute_deposit function, the contract adds two attributes to the response object, which provide information about the action that was performed and the recipient address of the tokens that were sent. However, these attributes are not emitted as events, and thus, users are unable to track these actions.

This vulnerability can potentially be exploited by attackers, as they can perform actions on the contract that are not tracked or recorded by the contract. This lack of transparency can also make it difficult for users to determine the cause of any issues or errors that may occur while interacting with the contract.

Recommendation: To address this vulnerability, it is recommended to emit events in all functions of the contract, including the execute_deposit function. This will provide transparency to users and allow them to track the actions performed by the contract. Additionally, emitting events can help with debugging and identifying any issues that may occur while interacting with the contract.

Status: Open

3. Potential Arithmetic Underflow in Withdraw Function

Description: The contract's withdraw function allows users to withdraw a specified amount from the jackpot account. However, the function does not check if the amount parameter passed to it is greater than the current balance of the jackpot account. If the amount parameter is greater than the current balance of the jackpot account, the subtraction record.jackpot - amount will result in an integer underflow, which can cause unexpected behavior or even a contract crash.

Recommendation: To prevent integer underflows in the withdraw function, it is recommended to add a check that ensures the amount parameter is not greater than the current balance of the jackpot account before attempting to subtract it. This can be achieved by adding a simple if statement that checks the balance of the jackpot account against the amount parameter. If the amount is greater than the balance, the function should throw an error or return a message to the user indicating that the withdrawal cannot be completed.

Status: Amended
The team has fixed the issue.

4. Inappropriate Address Validation in receiver Parameter

Description: The contract's receiver parameter is validated using the deps.api.addr_validate function. However, if the receiver parameter is a malformed or invalid address, the deps.api.addr_validate function will return an error, which could cause the contract to behave unexpectedly or crash. Additionally, the same vulnerability exists in the owner_set function, which also uses the deps.api.addr_validate function to validate the owner parameter.

Recommendation: To prevent errors and unexpected behavior in the contract, it is recommended to validate the receiver and owner parameters before passing them to the deps.api.addr_validate function

Status: Open

5. Direct access to the funds array

Description: In the execute_deposit function, there is a potential issue when accessing the info.funds array directly without checking if it's empty:

```
if info.funds[0].amount != config.amount {
   return Err(ContractError::Invalid {});
}
```

In this line, the code accesses the first element of info.funds (i.e., info.funds[0]). However, if the info.funds array is empty, trying to access the first element will cause a panic, leading to an abrupt termination of the contract execution.

Recommendation: To fix this issue, you should first check if the info.funds array is empty before accessing its elements:

```
if info.funds.is_empty() {
   return Err(ContractError::NoFunds {});
}
if info.funds[0].amount != config.amount {
   return Err(ContractError::Invalid {});
}
```

In this example, before accessing the info.funds array, we first check if it's empty using the is_empty() method. If it's empty, the function returns a custom ContractError::NoFunds error. This way, we can ensure that the array is not empty when accessing its elements, preventing a potential panic.

Status: Amended
The team has fixed the issue.

Informational

1. Inexistent migration functionality

Description: The contract does not have a migrate function, which means that it cannot be upgraded without redeploying a new instance of the contract. This could be problematic if the contract contains bugs or requires new features to be added. It would require a new instance of the contract to be deployed, and any state data would need to be manually transferred to the new contract instance.

Recommendation: It is recommended to include a migrate function in the contract to allow for upgrading the contract while keeping its state data intact. The migrate function should be designed to ensure that the state data is migrated seamlessly and without any loss or corruption.

Status: Acknowledged

Developer's Response:
As for the migration functionality is concerned that was out of scope for the requirements provided for development. That is why it isn't implemented."

Automated Test Results

Cargo Audit:

cargo-audit is a command-line utility which inspects Cargo.lock files and compares them against the RustSec Advisory Database, a community database of security vulnerabilities maintained by the Rust Secure Code Working Group

```
Crate:     crossbeam-deque
Version:   0.8.0
Title:     Data race in crossbeam-deque
Date:      2021-07-30
ID:        RUSTSEC-2021-0093
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0093
Solution:  Upgrade to >=0.7.4, <0.8.0 OR >=0.8.1

Crate:     rand_core
Version:   0.6.1
Title:     Incorrect check on buffer length when seeding RNGs
Date:      2021-02-12
ID:        RUSTSEC-2021-0023
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0023
Solution:  Upgrade to >=0.6.2

Crate:     remove_dir_all
Version:   0.5.3
Title:     Race Condition Enabling Link Following and Time-of-check Time-of-use
(TOCTOU)
Date:      2023-02-24
ID:        RUSTSEC-2023-0018
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0018
Solution:  Upgrade to >=0.8.0

Crate:     mach
Version:   0.3.2
Warning:   unmaintained
Title:     mach is unmaintained
Date:      2020-07-14
ID:        RUSTSEC-2020-0168
URL:       https://rustsec.org/advisories/RUSTSEC-2020-0168
```

```
Crate:     parity-wasm
Version:   0.42.1
Warning:   unmaintained
Title:     Crate `parity-wasm` deprecated by the author
Date:      2022-10-01
ID:        RUSTSEC-2022-0061
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0061

Crate:     cpufeatures
Version:   0.2.2
Warning:   yanked

Crate:     crossbeam-deque
Version:   0.8.0
Warning:   yanked

Crate:     crossbeam-epoch
Version:   0.9.1
Warning:   yanked

Crate:     crossbeam-utils
Version:   0.8.1
Warning:   yanked

Crate:     pin-project-lite
Version:   0.2.4
Warning:   yanked

Crate:     rand_core
Version:   0.6.1
Warning:   yanked
```

Cargo Tarpaulin:

Running cargo tarpaulin on the project code base results in panic showing that test quality , coverage and scope is insufficient.

```
|| Uncovered Lines:
|| src/contract.rs: 18, 25-28, 31-33, 36, 38-40, 44, 50-54, 58, 62-65, 67-72, 74-76, 78-79,
82, 89-92, 94-95, 97-99, 101-102, 105, 111-113, 116-118, 121-122, 125, 131-133, 136-139,
141-142, 146-150, 152-153, 157-161, 163-165, 167-170, 174-178, 182-188, 191-196, 199-202
|| Tested/Total Lines:
|| src/contract.rs: 0/106
||
0.00% coverage, 0/106 lines covered
```

Concluding Remarks

While conducting the audits of the Burnicorn, it was observed that the contracts contain Medium and Low severity issues along with a few recommendations.

Our auditors suggest that Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Burnicorn platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

IMMUNEBYTES

AUDITS