OrangeCat

Smart Contract Audit Report

IMMUNE BYTES

Audits

June 08, 2021

Introduction

1. About Orange Cat

Orange Cat Token (ticker: OCAT) is a community-based blockchain token on a mission to unite animal lovers worldwide, with a vision of donating time and space to safeguard the environment. By building and supporting animal/nature reserves, Orange Cat Token is dedicated to protecting both the natural habitat of our fuzzy friends as well as the interests of animal-lovers, by constantly improving transparency and continuity to achieve next-generation philanthropic goals.

Visit https://www.orangecat.info/ to learn more about.

2. About ImmuneBytes

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

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

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

Documentation Details

No documentation was provided by the OrangeCat team for the purpose of the audit.

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 safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any 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 in order 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.

The code was audited by a team of independent auditors which includes -

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: OrangeCat
- Languages: Solidity(Smart contract)
- Github commit hash/Smart Contract Address for audit:
    - Link: https://bscscan.com/address/0x7128e52ca302c5f5beeb801b6ad373fdebe3dc5e#code
- Testnet deployment
    - Orange Cat Token: 0x9650090d53bEee1106D9C6387e3372CD68831417
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

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 within 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

Security Level References

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

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	-	-	2
Closed	-	-	-

Admin/Owner Privileges

The **admin/owner** of **Orange Cat Token** smart contract has a privilege over the smart contract. This privilege can be misused either intentionally or unintentionally (in case admin's private key gets hacked). We assume that this extra right will always be used appropriately. The privilege is listed below.

1. **The owner of the Orange Cat Token contract can mint any amount of tokens.**
   The **Orange Cat Token** contract contains a **mint**() function by which the **owner** account can mint any number of **Orange Cat** tokens to his own address.

Recommendation:
Consider hardcoding predefined range or validations for input variable in privileged access function. Also consider adding some governance for admin rights for smart contract or use a multi-sig wallet as admin/owner address.

Low severity issues

1. **Inconsistent use of *msg.sender* and *_msgSender()*.**
   In the **BEP20Token** contract, throughout the contract **_msgSender()** function is used to access the caller's address while in its **constructor() msg.sender** is used. Both the methods return the same value but still it is always recommended to use the same coding conventions throughout the smart contract. Consistent coding conventions makes the code much more readable.

   Recommendation:
   Consider using any one of these - **msg.sender** or **_msgSender()**.

2. **Unused internal functions present.**
   In the **BEP20Token** smart contract, there are two **internal** functions implemented **_burn**() and **_burnFrom**() which are never used inside the smart contract. Every function in Solidity increases the size of bytecode for the smart contract so it is always recommended to remove any extra unused functions.

   Recommendation:
   Consider removing the unused functions.

Automated Auditing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to them in real-time. We performed analysis using contract Library on the Rinkeby address of the BEP20Token contract used during manual testing:

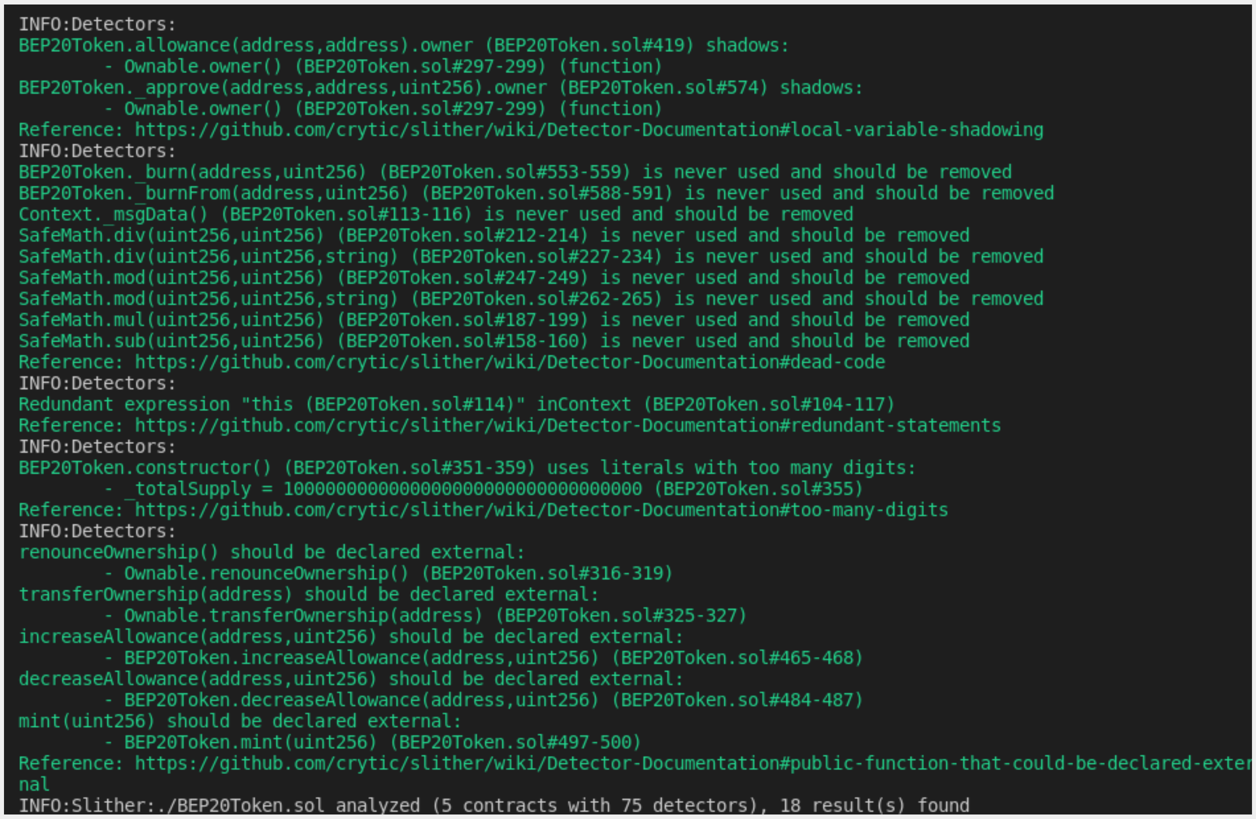
- Orange Cat Token: 0x9650090d53bEee1106D9C6387e3372CD68831417

It raises no major concern for the contracts.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit.



```
INFO:Detectors:
BEP20Token.allowance(address,address).owner (BEP20Token.sol#419) shadows:
        - Ownable.owner() (BEP20Token.sol#297-299) (function)
BEP20Token._approve(address,address,uint256).owner (BEP20Token.sol#574) shadows:
        - Ownable.owner() (BEP20Token.sol#297-299) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
BEP20Token._burn(address,uint256) (BEP20Token.sol#553-559) is never used and should be removed
BEP20Token._burnFrom(address,uint256) (BEP20Token.sol#588-591) is never used and should be removed
Context._msgData() (BEP20Token.sol#113-116) is never used and should be removed
SafeMath.div(uint256,uint256) (BEP20Token.sol#212-214) is never used and should be removed
SafeMath.div(uint256,uint256,string) (BEP20Token.sol#227-234) is never used and should be removed
SafeMath.mod(uint256,uint256) (BEP20Token.sol#247-249) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (BEP20Token.sol#262-265) is never used and should be removed
SafeMath.mul(uint256,uint256) (BEP20Token.sol#187-199) is never used and should be removed
SafeMath.sub(uint256,uint256) (BEP20Token.sol#158-160) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Redundant expression "this (BEP20Token.sol#114)" inContext (BEP20Token.sol#104-117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
BEP20Token.constructor() (BEP20Token.sol#351-359) uses literals with too many digits:
        - _totalSupply = 1000000000000000000000000000000000 (BEP20Token.sol#355)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (BEP20Token.sol#316-319)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (BEP20Token.sol#325-327)
increaseAllowance(address,uint256) should be declared external:
        - BEP20Token.increaseAllowance(address,uint256) (BEP20Token.sol#465-468)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20Token.decreaseAllowance(address,uint256) (BEP20Token.sol#484-487)
mint(uint256) should be declared external:
        - BEP20Token.mint(uint256) (BEP20Token.sol#497-500)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-exter
nal
INFO:Slither:./BEP20Token.sol analyzed (5 contracts with 75 detectors), 18 result(s) found
```

Concluding Remarks

While conducting the audits of the OrangeCat smart contract, it was observed that the contracts contain Low severity issues, along with a few areas of Admin/Owner privileges.

Our auditors suggest that Low severity issues should be resolved by the developers. Resolving the areas of recommendations are up to the team's discretion. 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 OrangeCat 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 the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.