Mozart Finance

Smart Contract Audit Report

IMMUNE BYTES

Audits

Mozart Finance

March 15, 2021

IMMUNE BYTES

Introduction

1. About Mozart Finance

Mozart Finance is a fork from Goose Finance & astonishing DeFi project running on Binance Smart Chain with lots of other features that let you earn and win tokens.

Mozart Finance is trying to create a splendid perpetual deflation token that performs like a symphony. Their native token PIANO will provide a stable price pump with a sufficient burn mechanism. They are not trying to replace the swap & exchange but rather to add value into our system and create a sustainable environment for people to yield farm with high APR and later even more than that.

Learn more about Mozart Finance here: https://mozartfinance.gitbook.io/mozart-finance/

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

Mozart Finance team has provided documentation for the purpose of conducting the audit. The documents are:

1. https://mozartfinance.gitbook.io/mozart-finance/

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: Mozart Finance
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck
- Github commits hash/Smart Contract Address for audit: https://bscscan.com/address/0x9eEC1044C5bD15782F806C63003F4730eeDfDAE4#code
- BscScan Code (Testnet):
    - Piano Token: 0x7507a112ac7532F7717f430106a8BcaE58f69E3a

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:

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	1	1
Closed	-	-	-

High severity issues

1. As specified in the documentation's feature section: https://mozartfinance.gitbook.io/mozart-finance/#features, every transaction will burn some amount of tokens. The required functionality is included in the transfer() function. But the required _burn() is excluded in the transferFrom() function. This means users transferring tokens via transferfrom will be able to move their tokens without burning them. It can be exploited by users as this is an external function.

   Recommendation:
   Implementation is included with the fix of 2nd high severity issue.

2. In the PianoTokens contract, delegates are not moved to the receiver's *delegates* mapping in transfer and transferFrom function.

   This will impact the vote count of delegates by the amount transferred as the number of votes added to the delegate's vote count is equivalent to the balance of PIANO tokens in the user's account. This means delegation won't be reduced even if the user votes and moves his PIANO tokens, allowing people to double-spend their voting power.

   Recommendation:
   To override transfer() and transferFrom() functions from BEP20 contract and implement them as follows:

```
function transfer(address _to, uint256 _value) public override
returns (bool success) {
       require(_value != 0, "PIANO::transfer: transfer value should
not be zero");
       uint256 toBurn = _value / 100;
       if(super.transfer(_to, _value - toBurn)) {
           _burn(msg.sender, toBurn);
           _moveDelegates(delegates[msg.sender], delegates[_to],
_value - toBurn);
           return true;
       }
       return false;
}
```

```
function transferFrom(address _from, address _to, uint256 _value)
public override returns (bool success) {
       require(_value != 0, "PIANO::transfer: transfer value should
not be zero");
       uint256 toBurn = _value / 100;
       if(super.transferFrom(_from, _to, _value - toBurn)) {
           _burn(msg.sender, toBurn);
           _moveDelegates(delegates[_from], delegates[_to], _value -
toBurn);
           return true;
       }
       return false;
}
```

Medium severity issues

1. There are 2 public functions declared for minting PIANO Tokens
   Once in BEP20 contract (Line number - 764)

```
function mint(uint256 amount) public onlyOwner returns (bool) {
       _mint(_msgSender(), amount);
       return true;
   }
```

   And another function in PianoToken contract (Line number - 876):

```
function mint(address _to, uint256 _amount) public onlyOwner {
       _mint(_to, _amount);
       _moveDelegates(address(0), _delegates[_to], _amount);
   }
```

   mint() function in BEP20 does not include _moveDelegates() implementation which is important in tracking the user's votes. Therefore a user whose tokens are minted using the BEP20 mint will not have voting power. This will create confusion among protocol token holders.

As these functions are onlyOwner (can only be accessed by the owner of the protocol), hence considered as a medium-level severity issue. As the risk is not exposed to users themselves and if the owner only uses the right function, issues could be avoided.

It is recommended to either remove the function from BEP20 and redeploy the contract or use the BEP20.mint() function carefully.

Low severity issues

1. It is recommended to remove the getOwner function to avoid gas wastage while deploying the contract.

   As redundant functions to get the owner of Piano Token were found in IBEP20 and BEP20 contracts:

```
function getOwner() external view returns (address);
```

   Ownable contract already contains following function to get the Owner:

```
function owner() public view returns (address) {
      return _owner;
}
```

Recommendations

1. A function with a ***public*** visibility modifier that is not called internally should be set to ***external*** visibility to increase code readability. Moreover, in many cases, functions with ***external*** visibility modifier spend less gas compared to functions with ***public*** visibility modifier. Following functions can be declared external:
   - ***renounceOwnership() (Piano.sol#93-96)***
   - ***transferOwnership(address) (Piano.sol#102-104)***
   - ***decimals() (Piano.sol#665-667)***
   - ***symbol() (Piano.sol#672-674)***
   - ***allowance(address,address) (Piano.sol#711-718)***
   - ***approve(address,uint256) (Piano.sol#727-734)***
   - ***transferFrom(address,address,uint256) (Piano.sol#748-763)***
   - ***increaseAllowance(address,uint256) (Piano.sol#777-787)***
   - ***decreaseAllowance(address,uint256) (Piano.sol#803-816)***
   - ***mint(uint256) (Piano.sol#826-829)***
   - ***mint(address,uint256) (Piano.sol#946-949)***
   - ***burn(uint256) (Piano.sol#951-956)***

2. In PianoToken, transfer() function is declared as virtual. (Line number - 666)

```
function transfer(address _to, uint256 _value) public virtual override returns (bool success) {}
```

   Base functions can be overridden by inheriting contracts to change their behaviour if they are marked as `virtual`. Hence it is recommended to remove the virtual keyword:

```
function transfer(address _to, uint256 _value) public override returns (bool success) {}
```

3. Compiler version for Piano Token contract is **>=0.4.0**

```
pragma solidity >=0.4.0;
```

   This contract is deployed at 0x9eEC1044C5bD15782F806C63003F4730eeDfDAE4 for the **0.6.12** version which could confuse users as there are a lot of breaking changes between 0.4.0 and 0.6.0 versions hence it is recommended to use a fixed compiler version in the smart contract (Ex: **pragma solidity 0.6.12;**). Also, this is not a security threat since the contract has already been deployed, hence included as a recommendation.

4. In PianoToken, multiple functions do not check for amounts. It is recommended to add a check for 0 amount.
    1. In mint() function (Line number - 876)

```
require(_amount != 0, "PIANO::mint: mint value should not be zero");
```

    2. In burn() function (Line number - 881)

```
require(value != 0, "PIANO::burn: burn value should not be zero");
```

    3. In transfer() function (Line number - 888)

```
require(_value != 0, "PIANO::transfer: transfer value should not be zero");
```

Unit Test

No unit tests were provided by the Mozart Finance team.

Recommendation:
Our team suggests that the developers should write extensive test cases for the contracts.

Coverage Report

Coverage report cannot be generated without unit test cases.

Recommendation:
We recommend 100% line and branch coverage for unit test cases.

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 contract.

SmartCheck

SmartCheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. SmartCheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false-negative rate (FNR). It gave the following result for the PianoToken contract:

- Piano Token: https://tool.smartdec.net/scan/004dd019cc294bdcb033a5afd7315b0f

SmartCheck did not detect any high severity issue. All the considerable issues raised by SmartCheck are already covered in the Manual Audit section of this report.

Contract Library

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

- Piano Token: 0x66e4AB2601299ACF045FF8890D45baDF0E20877c

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 section.

```
-> PianoToken

INFO:Detectors:
PianoToken._writeCheckpoint(address,uint32,uint256,uint256) (Piano.sol#1184-1210) uses a dangerous strict
equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber
(Piano.sol#1197-1198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BEP20.constructor(string,string).name (Piano.sol#642) shadows:
        - BEP20.name() (Piano.sol#658-660) (function)
        - IBEP20.name() (Piano.sol#140) (function)
BEP20.constructor(string,string).symbol (Piano.sol#642) shadows:
        - BEP20.symbol() (Piano.sol#672-674) (function)
        - IBEP20.symbol() (Piano.sol#135) (function)
BEP20.allowance(address,address).owner (Piano.sol#711) shadows:
        - Ownable.owner() (Piano.sol#74-76) (function)
BEP20._approve(address,address,uint256).owner (Piano.sol#914) shadows:
        - Ownable.owner() (Piano.sol#74-76) (function)
PianoToken.burn(uint256).totalSupply (Piano.sol#952) shadows:
        - BEP20.totalSupply() (Piano.sol#679-681) (function)
        - IBEP20.totalSupply() (Piano.sol#125) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
PianoToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (Piano.sol#1044-1083) uses
timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,PIANO::delegateBySig: signature expired) (Piano.sol#1081)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (Piano.sol#436-448) uses assembly
        - INLINE ASM (Piano.sol#444-446)
Address._functionCallWithValue(address,bytes,uint256,string) (Piano.sol#563-590) uses assembly
        - INLINE ASM (Piano.sol#582-585)
PianoToken.getChainId() (Piano.sol#1221-1227) uses assembly
        - INLINE ASM (Piano.sol#1223-1225)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version>=0.4.0 (Piano.sol#13) allows old versions
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Piano.sol#466-478):
        - (success) = recipient.call{value: amount}() (Piano.sol#473)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Piano.sol#563-590):
        - (success,returndata) = target.call{value: weiValue}(data) (Piano.sol#572-573)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PianoToken.mint(address,uint256)._to (Piano.sol#946) is not in mixedCase
Parameter PianoToken.mint(address,uint256)._amount (Piano.sol#946) is not in mixedCase
Parameter PianoToken.transfer(address,uint256)._to (Piano.sol#958) is not in mixedCase
Parameter PianoToken.transfer(address,uint256)._value (Piano.sol#958) is not in mixedCase
Variable PianoToken._delegates (Piano.sol#978) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Piano.sol#35)" inContext (Piano.sol#25-38)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
	- Ownable.renounceOwnership() (Piano.sol#93-96)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (Piano.sol#102-104)
decimals() should be declared external:
	- BEP20.decimals() (Piano.sol#665-667)
symbol() should be declared external:
	- BEP20.symbol() (Piano.sol#672-674)
allowance(address,address) should be declared external:
	- BEP20.allowance(address,address) (Piano.sol#711-718)
approve(address,uint256) should be declared external:
	- BEP20.approve(address,uint256) (Piano.sol#727-734)
transferFrom(address,address,uint256) should be declared external:
	- BEP20.transferFrom(address,address,uint256) (Piano.sol#748-763)
increaseAllowance(address,uint256) should be declared external:
	- BEP20.increaseAllowance(address,uint256) (Piano.sol#777-787)
decreaseAllowance(address,uint256) should be declared external:
	- BEP20.decreaseAllowance(address,uint256) (Piano.sol#803-816)
mint(uint256) should be declared external:
	- BEP20.mint(uint256) (Piano.sol#826-829)
mint(address,uint256) should be declared external:
	- PianoToken.mint(address,uint256) (Piano.sol#946-949)
burn(uint256) should be declared external:
	- PianoToken.burn(uint256) (Piano.sol#951-956)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (7 contracts with 72 detectors), 32 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Concluding Remarks

While conducting the audits of Mozart Finance smart contract, it was observed that the contracts contain High, Medium, and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High, Medium, Low severity issues should be resolved by Mozart Finance 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 Mozart Finance platform or its product neither this audit is investment advice.
Notes:

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

ImmuneBytes Pvt Ltd.