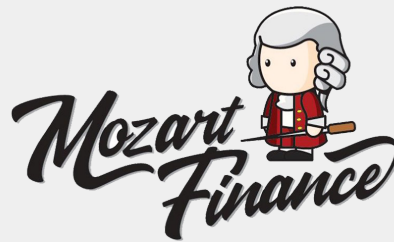


Mozart Finance

MasterChef

Smart Contract Audit Report



March 27, 2021

Introduction	3
About Mozart Finance	3
About ImmuneBytes	3
Documentation Details	3
Audit Process & Methodology	4
Audit Details	4
Audit Goals	5
Security Level References	5
High severity issues	6
Low severity issues	6
Recommendations	7
Unit Test	8
Automated Auditing	9
Solhint Linting Violations	9
Contract Library	9
Slither	9
Concluding Remarks	17
Disclaimer	17

This 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.

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/>

This 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.

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://testnet.bscscan.com/address/0xf5d07Bc0c404f5894B65d5D4F599e3E419077948#code>
- BscScan Code (Testnet):
MasterChef:
<https://testnet.bscscan.com/address/0xC7d4eCCed69DF812C4e9F0Af259498189C7540B4>

This 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.

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	<u>High</u>	<u>Medium</u>	<u>Low</u>
Open	-	-	-
Closed	1	-	1

This 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.

High severity issues

1. The following variable initialization in constructor have wrong values from what is expected from the names.

```
dayTimestamp = now + 10 minutes;
firstWeekTimestamp = now + 15 minutes;
secondWeekTimestamp = now + 20 minutes;
thirdWeekTimestamp = now + 25 minutes;
```

They should be updated to the following values if the intention is the same as the names declared. Otherwise, names should be updated to clear the intentions.

```
dayTimestamp = now + 1 day;
firstWeekTimestamp = now + 1 week;
secondWeekTimestamp = now + 2 week;
thirdWeekTimestamp = now + 3 week;
```

Amended (March 27th 2021): Issue was fixed by Mozart Finance team and is no longer present in:

<https://bscscan.com/address/0x2AED73484814e67F8b795Fc38Af71E2CEc429d0f>

Low severity issues

1. Mint function calls should check for zero amounts to avoid reverting the transaction when rewards are reduced to zero in the first case and when foam minted is 0 in enterStaking() function. This will also reduce gas consumed in cross contract calls.

MasterChef.updatePool(uint256)

```
if (pianoReward > 0) {
    piano.mint(devaddr, pianoReward.div(10));
    piano.mint(address(foam), pianoReward);
}
```

MasterChef.enterStaking(uint256)

```
if (_amount > 0) foam.mint(msg.sender, _amount);
```

Amended (March 27th 2021): Issue was fixed by Mozart Finance team and is no longer present in:

<https://bscscan.com/address/0x2AED73484814e67F8b795Fc38Af71E2CEc429d0f>

This 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.

Recommendations

1. Compiler version for the contract is **0.6.12**. It is recommended to use the latest solidity version to include potential gas and security improvements. This is not a security issue, but just a recommendation to improve the product.
2. 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:
 - **updateMultiplier(uint256) (MasterChef.sol#1641-1643)**
 - **add(uint256,IBEP20,uint16,bool) (MasterChef.sol#1651-1666)**
 - **set(uint256,uint256,uint16,bool) (MasterChef.sol#1669-1681)**
 - **getPerBlock() (MasterChef.sol#1726-1728)**
 - **deposit(uint256,uint256) (MasterChef.sol#1759-1791)**
 - **withdraw(uint256,uint256) (MasterChef.sol#1794-1812)**
 - **enterStaking(uint256) (MasterChef.sol#1815-1833)**
 - **leaveStaking(uint256) (MasterChef.sol#1836-1853)**
 - **emergencyWithdraw(uint256) ((MasterChef.sol#1856-1863)**
 - **dev(address) (MasterChef.sol#1871-1874)**
3. The use of the SafeMath library can be improved as suggested below to reduce gas consumption.

The following lines have used the div() function from the SafeMath library. It can be removed as the divisor is non-zero in these cases hence we can save the gas.

```
Line#1777
uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);

Line#1781
pool.lpToken.safeTransfer(pianoaddr,depositFee.div(4).div(4).mul(3))
;
```

It is recommended to use multiplications before division operations to reduce precision errors.

```
Line#1781
pool.lpToken.safeTransfer(pianoaddr,depositFee.div(4).div(4).mul(3))
;
```

This 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.

It is recommended to remove SafeMath functions from below as depositFee is in percentage whose value is checked already hence this will not cause overflow/underflow situations.

```
Line#1784
user.amount = user.amount.add(_amount).sub(depositFee);
```

It is recommended to remove the following safeTransfer() function because the token is transferred to the current contract where tokens are already present. Hence making it a useless transaction.

```
Line#1780
pool.lpToken.safeTransfer(address(this), depositFee.div(4));
```

4. The function *getPerBlock()* is not required as the variable is already public hence it can be accessed in place of a getter.

```
function getPerBlock() public view returns (uint256) {
    return updatedPianoPerBlock;
}
```

Unit Test

```
> PancakeSwap@1.0.0 test
> npx builder test
```

You probably meant to type buidler. We got you.
All contracts have already been compiled, skipping compilation.

```
Contract: MasterChef
  ✓ real case (1313ms)
  ✓ deposit/withdraw (305ms)
  ✓ staking/unstaking (221ms)
  ✓ update multiplier (483ms)
  ✓ should allow dev and only dev to update dev (56ms)
```

```
5 passing (4s)
```

This 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.

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.

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 the contract Library on the Kovan address of the MasterChef contract, used during manual testing:

- MasterChef: [0xf755B4b6694771fF1Db30248A198FCE3EC8e20Bc](#)

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.

```
INFO:Detectors:
MasterChef (MasterChef.sol#1544-1876) contract sets array length with a user-controlled value:
- poolInfo.push(PoolInfo(_piano,1000,startBlock,0,0)) (MasterChef.sol#1629-1635)
MasterChef (MasterChef.sol#1544-1876) contract sets array length with a user-controlled value:
- poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP))
(MasterChef.sol#1658-1664)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#array-length-assignment
INFO:Detectors:
MasterChef.pendingPiano(uint256,address) (MasterChef.sol#1702-1724) performs a multiplication on the result of
a division:
-pianoReward = multiplier.mul(updatedPianoPerBlock).mul(pool.allocPoint).div(totalAllocPoint)
(MasterChef.sol#1720)
-accPianoPerShare = accPianoPerShare.add(pianoReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1721)
MasterChef.updatePool(uint256) (MasterChef.sol#1740-1756) performs a multiplication on the result of a division:
-pianoReward = multiplier.mul(pianoPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#1751)
-pool.accPianoPerShare = pool.accPianoPerShare.add(pianoReward.mul(1e12).div(lpSupply))
(MasterChef.sol#1754)
```

This 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.

MasterChef.deposit(uint256,uint256) (MasterChef.sol#1759-1791) performs a multiplication on the result of a division:

- pool.lpToken.safeTransfer(pianoaddr,depositFee.div(4).div(4).mul(3)) (MasterChef.sol#1781)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply>

INFO:Detectors:

PianoToken._writeCheckpoint(address,uint32,uint256,uint256) (MasterChef.sol#1224-1242) uses a dangerous strict equality:

- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (MasterChef.sol#1234)

FoamBar._writeCheckpoint(address,uint32,uint256,uint256) (MasterChef.sol#1493-1511) uses a dangerous strict equality:

- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (MasterChef.sol#1503)

MasterChef.updatePool(uint256) (MasterChef.sol#1740-1756) uses a dangerous strict equality:

- lpSupply == 0 (MasterChef.sol#1746)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities>

INFO:Detectors:

Reentrancy in MasterChef.add(uint256,IBEP20,uint16,bool) (MasterChef.sol#1651-1666):

External calls:

- massUpdatePools() (MasterChef.sol#1654)
 - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
 - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)

State variables written after the call(s):

- poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP)) (MasterChef.sol#1658-1664)
- updateStakingPool() (MasterChef.sol#1665)
 - poolInfo[0].allocPoint = points (MasterChef.sol#1692)
- totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChef.sol#1657)
- updateStakingPool() (MasterChef.sol#1665)
 - totalAllocPoint = totalAllocPoint.sub(poolInfo[0].allocPoint).add(points) (MasterChef.sol#1691)

Reentrancy in MasterChef.deposit(uint256,uint256) (MasterChef.sol#1759-1791):

External calls:

- updatePool(_pid) (MasterChef.sol#1767)
 - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
 - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1771)
 - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1775)
- pool.lpToken.safeTransfer(treasuryaddr,depositFee.div(2)) (MasterChef.sol#1779)
- pool.lpToken.safeTransfer(address(this),depositFee.div(4)) (MasterChef.sol#1780)
- pool.lpToken.safeTransfer(pianoaddr,depositFee.div(4).div(4).mul(3)) (MasterChef.sol#1781)
- pool.lpToken.safeTransfer(devaddr,depositFee.div(4).div(4)) (MasterChef.sol#1782)

State variables written after the call(s):

- user.amount = user.amount.add(_amount).sub(depositFee) (MasterChef.sol#1784)

Reentrancy in MasterChef.deposit(uint256,uint256) (MasterChef.sol#1759-1791):

External calls:

- updatePool(_pid) (MasterChef.sol#1767)
 - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
 - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1771)
 - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1775)

State variables written after the call(s):

- user.amount = user.amount.add(_amount) (MasterChef.sol#1786)

Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1856-1863):

External calls:

- pool.lpToken.safeTransfer(address(msg.sender),user.amount) (MasterChef.sol#1859)

This 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.

```

State variables written after the call(s):
- user.amount = 0 (MasterChef.sol#1861)
- user.rewardDebt = 0 (MasterChef.sol#1862)
Reentrancy in MasterChef.enterStaking(uint256) (MasterChef.sol#1815-1833):
  External calls:
  - updatePool(0) (MasterChef.sol#1818)
    - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
    - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
  - safePianoTransfer(msg.sender,pending) (MasterChef.sol#1822)
    - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
  - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1826)
  State variables written after the call(s):
  - user.amount = user.amount.add(_amount) (MasterChef.sol#1827)
  - user.rewardDebt = user.amount.mul(pool.accPianoPerShare).div(1e12) (MasterChef.sol#1829)
Reentrancy in MasterChef.leaveStaking(uint256) (MasterChef.sol#1836-1853):
  External calls:
  - updatePool(0) (MasterChef.sol#1840)
    - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
    - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
  - safePianoTransfer(msg.sender,pending) (MasterChef.sol#1843)
    - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
  State variables written after the call(s):
  - user.amount = user.amount.sub(_amount) (MasterChef.sol#1846)
Reentrancy in MasterChef.leaveStaking(uint256) (MasterChef.sol#1836-1853):
  External calls:
  - updatePool(0) (MasterChef.sol#1840)
    - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
    - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
  - safePianoTransfer(msg.sender,pending) (MasterChef.sol#1843)
    - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
  - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1847)
  State variables written after the call(s):
  - user.rewardDebt = user.amount.mul(pool.accPianoPerShare).div(1e12) (MasterChef.sol#1849)
Reentrancy in MasterChef.set(uint256,uint256,uint16,bool) (MasterChef.sol#1669-1681):
  External calls:
  - massUpdatePools() (MasterChef.sol#1672)
    - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
    - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
  State variables written after the call(s):
  - poolInfo[_pid].allocPoint = _allocPoint (MasterChef.sol#1675)
  - poolInfo[_pid].depositFeeBP = _depositFeeBP (MasterChef.sol#1676)
  - updateStakingPool() (MasterChef.sol#1679)
    - poolInfo[0].allocPoint = points (MasterChef.sol#1692)
  - totalAllocPoint = totalAllocPoint.sub(prevAllocPoint).add(_allocPoint) (MasterChef.sol#1678)
  - updateStakingPool() (MasterChef.sol#1679)
    - totalAllocPoint = totalAllocPoint.sub(poolInfo[0].allocPoint).add(points) (MasterChef.sol#1691)
Reentrancy in MasterChef.updatePool(uint256) (MasterChef.sol#1740-1756):
  External calls:
  - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
  - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
  State variables written after the call(s):
  - pool.accPianoPerShare = pool.accPianoPerShare.add(pianoReward.mul(1e12).div(lpSupply))
  (MasterChef.sol#1754)
  - pool.lastRewardBlock = block.number (MasterChef.sol#1755)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1794-1812):
  External calls:
  - updatePool(_pid) (MasterChef.sol#1801)

```

This 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.

```

- piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
- piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1804)
  - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
State variables written after the call(s):
- user.amount = user.amount.sub(_amount) (MasterChef.sol#1807)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1794-1812):
External calls:
- updatePool(_pid) (MasterChef.sol#1801)
  - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
  - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1804)
  - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1808)
State variables written after the call(s):
- user.rewardDebt = user.amount.mul(pool.accPianoPerShare).div(1e12) (MasterChef.sol#1810)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
FoamBar.safePianoTransfer(address,uint256) (MasterChef.sol#1287-1294) ignores return value by
piano.transfer(_to,pianoBal) (MasterChef.sol#1290)
FoamBar.safePianoTransfer(address,uint256) (MasterChef.sol#1287-1294) ignores return value by
piano.transfer(_to,_amount) (MasterChef.sol#1292)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
BEP20.constructor(string,string).name (MasterChef.sol#714) shadows:
- BEP20.name() (MasterChef.sol#730-732) (function)
- IBEP20.name() (MasterChef.sol#415) (function)
BEP20.constructor(string,string).symbol (MasterChef.sol#714) shadows:
- BEP20.symbol() (MasterChef.sol#744-746) (function)
- IBEP20.symbol() (MasterChef.sol#410) (function)
BEP20.allowance(address,address).owner (MasterChef.sol#778) shadows:
- Ownable.owner() (MasterChef.sol#621-623) (function)
BEP20._approve(address,address,uint256).owner (MasterChef.sol#950) shadows:
- Ownable.owner() (MasterChef.sol#621-623) (function)
PianoToken.burn(uint256).totalSupply (MasterChef.sol#993) shadows:
- BEP20.totalSupply() (MasterChef.sol#751-753) (function)
- IBEP20.totalSupply() (MasterChef.sol#400) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MasterChef.constructor(PianoToken,FoamBar,address,address,uint256,uint256,uint256)._devaddr
(MasterChef.sol#1609) lacks a zero-check on :
- devaddr = _devaddr (MasterChef.sol#1617)
MasterChef.constructor(PianoToken,FoamBar,address,address,uint256,uint256,uint256)._treasuryaddr
(MasterChef.sol#1610) lacks a zero-check on :
- treasuryaddr = _treasuryaddr (MasterChef.sol#1618)
MasterChef.dev(address)._devaddr (MasterChef.sol#1871) lacks a zero-check on :
- devaddr = _devaddr (MasterChef.sol#1873)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256) (MasterChef.sol#1759-1791):
External calls:
- updatePool(_pid) (MasterChef.sol#1767)
  - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
  - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1771)
  - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1775)

```

This 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.

```

- pool.lptoken.safeTransfer(treasuryaddr,depositFee.div(2)) (MasterChef.sol#1779)
- pool.lptoken.safeTransfer(address(this),depositFee.div(4)) (MasterChef.sol#1780)
- pool.lptoken.safeTransfer(pianoaddr,depositFee.div(4).div(4).mul(3)) (MasterChef.sol#1781)
- pool.lptoken.safeTransfer(devaddr,depositFee.div(4).div(4)) (MasterChef.sol#1782)
Event emitted after the call(s):
- Deposit(msg.sender,_pid,_amount) (MasterChef.sol#1790)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1856-1863):
External calls:
- pool.lptoken.safeTransfer(address(msg.sender),user.amount) (MasterChef.sol#1859)
Event emitted after the call(s):
- EmergencyWithdraw(msg.sender,_pid,user.amount) (MasterChef.sol#1860)
Reentrancy in MasterChef.enterStaking(uint256) (MasterChef.sol#1815-1833):
External calls:
- updatePool(0) (MasterChef.sol#1818)
  - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
  - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1822)
  - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lptoken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1826)
- foam.mint(msg.sender,_amount) (MasterChef.sol#1831)
Event emitted after the call(s):
- Deposit(msg.sender,0,_amount) (MasterChef.sol#1832)
Reentrancy in MasterChef.leaveStaking(uint256) (MasterChef.sol#1836-1853):
External calls:
- updatePool(0) (MasterChef.sol#1840)
  - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
  - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1843)
  - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lptoken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1847)
- foam.burn(msg.sender,_amount) (MasterChef.sol#1851)
Event emitted after the call(s):
- Withdraw(msg.sender,0,_amount) (MasterChef.sol#1852)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1794-1812):
External calls:
- updatePool(_pid) (MasterChef.sol#1801)
  - piano.mint(devaddr,pianoReward.div(10)) (MasterChef.sol#1752)
  - piano.mint(address(foam),pianoReward) (MasterChef.sol#1753)
- safePianoTransfer(msg.sender,pending) (MasterChef.sol#1804)
  - foam.safePianoTransfer(_to,_amount) (MasterChef.sol#1867)
- pool.lptoken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1808)
Event emitted after the call(s):
- Withdraw(msg.sender,_pid,_amount) (MasterChef.sol#1811)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
PianoToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (MasterChef.sol#1090-1131) uses
timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now <= expiry,PIANO::delegateBySig: signature expired) (MasterChef.sol#1129)
FoamBar.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (MasterChef.sol#1359-1400) uses
timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now <= expiry,PIANO::delegateBySig: signature expired) (MasterChef.sol#1398)
MasterChef.pendingPiano(uint256,address) (MasterChef.sol#1702-1724) uses timestamp for comparisons
Dangerous comparisons:
- now < dayTimestamp (MasterChef.sol#1709)
- dayTimestamp < now && now < firstWeekTimestamp (MasterChef.sol#1711)

```

This 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.


```

- firstWeekTimestamp < now && now < secondWeekTimestamp (MasterChef.sol#1713)
- now > secondWeekTimestamp (MasterChef.sol#1715)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (MasterChef.sol#257-268) uses assembly
- INLINE ASM (MasterChef.sol#264-266)
Address._functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#365-391) uses assembly
- INLINE ASM (MasterChef.sol#383-386)
PianoToken.getChainId() (MasterChef.sol#1249-1253) uses assembly
- INLINE ASM (MasterChef.sol#1251)
FoamBar.getChainId() (MasterChef.sol#1518-1522) uses assembly
- INLINE ASM (MasterChef.sol#1520)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used in :
- Version used: ['0.6.12', '>=0.4.0']
- >=0.4.0 (MasterChef.sol#21)
- 0.6.12 (MasterChef.sol#979)
- 0.6.12 (MasterChef.sol#1259)
- 0.6.12 (MasterChef.sol#1527)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version>=0.4.0 (MasterChef.sol#21) allows old versions
Pragma version0.6.12 (MasterChef.sol#979) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (MasterChef.sol#1259) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
Pragma version0.6.12 (MasterChef.sol#1527) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
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) (MasterChef.sol#286-292):
- (success) = recipient.call{value: amount}() (MasterChef.sol#290)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#365-391):
- (success,returndata) = target.call{value: weiValue}(data) (MasterChef.sol#374)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PianoToken.mint(address,uint256)._to (MasterChef.sol#985) is not in mixedCase
Parameter PianoToken.mint(address,uint256)._amount (MasterChef.sol#985) is not in mixedCase
Parameter PianoToken.transfer(address,uint256)._to (MasterChef.sol#999) is not in mixedCase
Parameter PianoToken.transfer(address,uint256)._value (MasterChef.sol#999) is not in mixedCase
Parameter PianoToken.transferFrom(address,address,uint256)._from (MasterChef.sol#1012) is not in mixedCase
Parameter PianoToken.transferFrom(address,address,uint256)._to (MasterChef.sol#1012) is not in mixedCase
Parameter PianoToken.transferFrom(address,address,uint256)._value (MasterChef.sol#1012) is not in mixedCase
Variable PianoToken._delegates (MasterChef.sol#1032) is not in mixedCase
Parameter FoamBar.mint(address,uint256)._to (MasterChef.sol#1266) is not in mixedCase
Parameter FoamBar.mint(address,uint256)._amount (MasterChef.sol#1266) is not in mixedCase
Parameter FoamBar.burn(address,uint256)._from (MasterChef.sol#1271) is not in mixedCase
Parameter FoamBar.burn(address,uint256)._amount (MasterChef.sol#1271) is not in mixedCase
Parameter FoamBar.safePianoTransfer(address,uint256)._to (MasterChef.sol#1287) is not in mixedCase
Parameter FoamBar.safePianoTransfer(address,uint256)._amount (MasterChef.sol#1287) is not in mixedCase
Variable FoamBar._delegates (MasterChef.sol#1302) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._allocPoint (MasterChef.sol#1651) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._lpToken (MasterChef.sol#1651) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._depositFeeBP (MasterChef.sol#1651) is not in mixedCase

```

This 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.

Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._withUpdate (MasterChef.sol#1651) is not in mixedCase
 Parameter MasterChef.set(uint256,uint256,uint16,bool)._pid (MasterChef.sol#1669) is not in mixedCase
 Parameter MasterChef.set(uint256,uint256,uint16,bool)._allocPoint (MasterChef.sol#1669) is not in mixedCase
 Parameter MasterChef.set(uint256,uint256,uint16,bool)._depositFeeBP (MasterChef.sol#1669) is not in mixedCase
 Parameter MasterChef.set(uint256,uint256,uint16,bool)._withUpdate (MasterChef.sol#1669) is not in mixedCase
 Parameter MasterChef.getMultiplier(uint256,uint256)._from (MasterChef.sol#1697) is not in mixedCase
 Parameter MasterChef.getMultiplier(uint256,uint256)._to (MasterChef.sol#1697) is not in mixedCase
 Parameter MasterChef.pendingPiano(uint256,address)._pid (MasterChef.sol#1702) is not in mixedCase
 Parameter MasterChef.pendingPiano(uint256,address)._user (MasterChef.sol#1702) is not in mixedCase
 Parameter MasterChef.updatePool(uint256)._pid (MasterChef.sol#1740) is not in mixedCase
 Parameter MasterChef.deposit(uint256,uint256)._pid (MasterChef.sol#1759) is not in mixedCase
 Parameter MasterChef.deposit(uint256,uint256)._amount (MasterChef.sol#1759) is not in mixedCase
 Parameter MasterChef.withdraw(uint256,uint256)._pid (MasterChef.sol#1794) is not in mixedCase
 Parameter MasterChef.withdraw(uint256,uint256)._amount (MasterChef.sol#1794) is not in mixedCase
 Parameter MasterChef.enterStaking(uint256)._amount (MasterChef.sol#1815) is not in mixedCase
 Parameter MasterChef.leaveStaking(uint256)._amount (MasterChef.sol#1836) is not in mixedCase
 Parameter MasterChef.emergencyWithdraw(uint256)._pid (MasterChef.sol#1856) is not in mixedCase
 Parameter MasterChef.safePianoTransfer(address,uint256)._to (MasterChef.sol#1866) is not in mixedCase
 Parameter MasterChef.safePianoTransfer(address,uint256)._amount (MasterChef.sol#1866) is not in mixedCase
 Parameter MasterChef.dev(address)._devaddr (MasterChef.sol#1871) is not in mixedCase
 Variable MasterChef.BONUS_MULTIPLIER (MasterChef.sol#1585) is not in mixedCase
 Reference:
<https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>
 INFO:Detectors:
 Redundant expression "this (MasterChef.sol#229)" inContext (MasterChef.sol#219-232)
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements>
 INFO:Detectors:
 renounceOwnership() should be declared external:
 - Ownable.renounceOwnership() (MasterChef.sol#640-643)
 transferOwnership(address) should be declared external:
 - Ownable.transferOwnership(address) (MasterChef.sol#649-651)
 decimals() should be declared external:
 - BEP20.decimals() (MasterChef.sol#737-739)
 symbol() should be declared external:
 - BEP20.symbol() (MasterChef.sol#744-746)
 allowance(address,address) should be declared external:
 - BEP20.allowance(address,address) (MasterChef.sol#778-780)
 approve(address,uint256) should be declared external:
 - BEP20.approve(address,uint256) (MasterChef.sol#789-792)
 increaseAllowance(address,uint256) should be declared external:
 - BEP20.increaseAllowance(address,uint256) (MasterChef.sol#832-835)
 decreaseAllowance(address,uint256) should be declared external:
 - BEP20.decreaseAllowance(address,uint256) (MasterChef.sol#851-858)
 mint(address,uint256) should be declared external:
 - PianoToken.mint(address,uint256) (MasterChef.sol#985-989)
 burn(uint256) should be declared external:
 - PianoToken.burn(uint256) (MasterChef.sol#991-997)
 mint(address,uint256) should be declared external:
 - FoamBar.mint(address,uint256) (MasterChef.sol#1266-1269)
 burn(address,uint256) should be declared external:
 - FoamBar.burn(address,uint256) (MasterChef.sol#1271-1274)
 safePianoTransfer(address,uint256) should be declared external:
 - FoamBar.safePianoTransfer(address,uint256) (MasterChef.sol#1287-1294)
 updateMultiplier(uint256) should be declared external:
 - MasterChef.updateMultiplier(uint256) (MasterChef.sol#1641-1643)
 add(uint256,IBEP20,uint16,bool) should be declared external:

This 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.

```

- MasterChef.add(uint256,IBEP20,uint16,bool) (MasterChef.sol#1651-1666)
set(uint256,uint256,uint16,bool) should be declared external:
- MasterChef.set(uint256,uint256,uint16,bool) (MasterChef.sol#1669-1681)
getPerBlock() should be declared external:
- MasterChef.getPerBlock() (MasterChef.sol#1726-1728)
deposit(uint256,uint256) should be declared external:
- MasterChef.deposit(uint256,uint256) (MasterChef.sol#1759-1791)
withdraw(uint256,uint256) should be declared external:
- MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1794-1812)
enterStaking(uint256) should be declared external:
- MasterChef.enterStaking(uint256) (MasterChef.sol#1815-1833)
leaveStaking(uint256) should be declared external:
- MasterChef.leaveStaking(uint256) (MasterChef.sol#1836-1853)
emergencyWithdraw(uint256) should be declared external:
- MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1856-1863)
dev(address) should be declared external:
- MasterChef.dev(address) (MasterChef.sol#1871-1874)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (10 contracts with 72 detectors), 112 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(MasterChef), it was observed that the contracts contain High and Low severity issues, along with several areas of recommendations.

Our auditors suggest that High and 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.

Note: Mozart Finance team has fixed the High and Low issues based on the auditor's recommendation.

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.