Security Assessment

moneytime-farm

Aug 14th, 2021

Table of Contents

Summary

This report has been prepared for Moneytime Finance to discover issues and vulnerabilities in the source code of the moneytime-farm project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	moneytime-farm
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/moneytimefinance/moneytime-farm
Commit	1.8675e643e0e97c621b922edb0f8d983b0b22863c 2.730fb209778166986bddf942c8b678e343adab97

Audit Summary

Delivery Date	Aug 14, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
Major	22	0	0	4	0	18
Medium	1	0	0	0	0	1
Minor	9	0	0	6	0	3
Informational	35	0	0	4	2	29
Discussion	0	0	0	0	0	0

Audit Scope

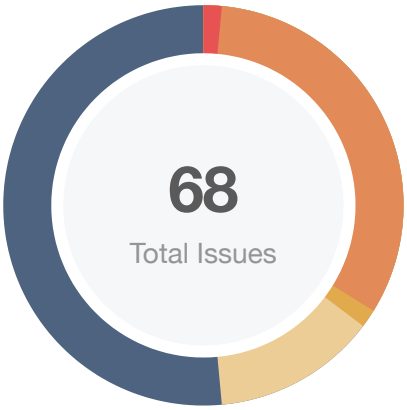
ID	File	SHA256 Checksum
MCM	MasterChefMoney.sol	4a894cf19cbaddae72e9f99666db5f4bc0534b01b319726449d08747529fd641
MCT	MasterChefTime.sol	70f414125c2624b2fc900781dde0a74ff7168ecf604f142f5d22f7e11cee230a
MTC	MoneyToken.sol	8ad792e5979e196c395d6bcd76435443fe3d295e5353f22d73e2188d2f4652ef
TTC	TimeToken.sol	8b307c99b6770cbfc237545f816913e819bcd8813fae886e48d5749ac1e610a0
TCK	Timelock.sol	202761ec32b607ae8625fd5c6027aed8cafce80d204c032ec14a021074aa14a1

It should be noted that the system design includes a number of economic arguments and assumptions. These were explored to the extent that they clarified the intention of the code base, but we did not audit the mechanism design itself.

Additionally, financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. The accuracy of the financial model is not in the scope of the audit.

Due to the large number of issues currently found, further testing and auditing are recommended to ensure the safety of this project.

Findings



Critical	**1** (1.47%)
Major	**22** (32.35%)
Medium	**1** (1.47%)
Minor	**9** (13.24%)
Informational	**35** (51.47%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
AAL-01	Unlocked Compiler Version & Version Inconsistency	Language Specific	Informational	Resolved
IBE-01	Unlocked Compiler Version & Version Inconsistency	Language Specific	Informational	Resolved
MCK-01	Unlocked Compiler Version & Version Inconsistency	Language Specific	Informational	Resolved
MCK-02	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved
MCM-01	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved
MCM-02	Set `constant` to Variables	Gas Optimization	Informational	Resolved
MCM-03	Missing `indexed` in Events	Gas Optimization	Informational	Resolved
MCM-04	Lack of Pool Validity Checks	Logical Issue	Informational	Resolved
MCM-05	Set `immutable` to Variables	Gas Optimization	Informational	Resolved
MCM-06	Comparison to A Boolean Constant	Gas Optimization	Informational	Resolved
MCM-07	Missing Emit Events	Gas Optimization	Informational	Resolved
MCM-08	`add()` Function Not Restricted	Volatile Code	Major	Acknowledged
MCM-09	Over Minted Token	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
MCM-10	Privileged Ownership	**Centralization / Privilege**	**Major**	Acknowledged
MCM-11	Incorrect Naming Convention Utilization	Coding Style	Informational	Partially Resolved
MCM-12	The Logical Issue of `depositFor()`	Logical Issue	Critical	Resolved
MCM-13	Calculation of `totalBusdAllocPoint`	Logical Issue	Major	Resolved
MCM-14	Calling Function `userIndex` Before Balance Updating	Control Flow	Medium	Resolved
MCM-15	Calculation of `busdPendingReward`	Logical Issue	Major	Resolved
MCM-16	`user.busdRewardDebt` Not Updated	Logical Issue	Major	Resolved
MCM-17	Lack of Input Validation	Volatile Code	Minor	Resolved
MCM-18	The Logic Issue of `add()`	Logical Issue	Informational	Resolved
MCM-19	The logical Issue of `emergencyWithdraw ()`	Logical Issue	Minor	Acknowledged
MCM-20	Token Transfer In `updateEmissionSettings`	**Centralization / Privilege**	**Major**	Resolved
MCM-21	The Logic Issue Of UnDistributed Rewards	Logical Issue	Informational	Acknowledged
MCM-22	Set The `secondaryReward`	Logical Issue	Informational	Acknowledged
MCM-23	Update of the `rewardDebt` and `busdRewardDebt` in `depositFor()`	Logical Issue	Major	Resolved
MCM-24	The logic of `payOrLockupPendingMoney()`	Logical Issue	Major	Resolved
MCT-01	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved
MCT-02	Set `immutable` to Variables	Gas Optimization	Informational	Resolved
MCT-03	Set `constant` to Variables	Gas Optimization	Informational	Resolved
MCT-04	Missing Emit Events	Gas Optimization	Informational	Resolved
MCT-05	Missing `indexed` in Events	Gas Optimization	Informational	Resolved

ID	Title	Category	Severity	Status
MCT-06	`add()` Function Not Restricted	Volatile Code	Major	Resolved
MCT-07	Check Effect Interaction Pattern Violated	Logical Issue	Minor	Resolved
MCT-08	Incompatibility With Deflationary Tokens	Logical Issue	Major	Resolved
MCT-09	Privileged Ownership	**Centralization / Privilege**	**Major**	Acknowledged
MCT-10	Incorrect Naming Convention Utilization	Coding Style	Informational	Partially Resolved
MCT-11	Lack of Pool Validity Checks	Logical Issue	Informational	Resolved
MCT-12	Comparison to A Boolean Constant	Gas Optimization	Informational	Resolved
MCT-13	Over Minted Token	Logical Issue	Minor	Acknowledged
MTC-01	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved
MTC-02	Lack of Input Validation	Volatile Code	Minor	Resolved
MTC-03	Delegation Not Moved Along With `transfer()` and `transferFrom()`	Logical Issue	Major	Resolved
MTC-04	Typos In The Contract	Coding Style	Informational	Resolved
MTC-05	Contract Gains Non-withdrawable ETH Via The `swapAndLiquify` Function	Logical Issue	Major	Resolved
MTC-06	Return Value Not Handled	Volatile Code	Informational	Resolved
MTC-07	Centralized Risk In `addLiquidity`	**Centralization / Privilege**	**Major**	Resolved
MTC-08	Third Party Dependencies	Control Flow	Minor	Acknowledged
MTC-09	Missing Emit Events	Gas Optimization	Informational	Resolved
MTC-10	Privileged Ownership	**Centralization / Privilege**	**Minor**	Acknowledged
MTC-11	Comparison to A Boolean Constant	Gas Optimization	Informational	Resolved

ID	Title	Category	Severity	Status
MTC-12	Centralized Risk In `safeTransfer()`	**Centralization / Privilege**	**Major**	Resolved
MTC-13	Centralized Risk In `safeTransferFrom()`	**Centralization / Privilege**	**Major**	Resolved
MTC-14	Centralized Risk In `claimBNB()`	**Centralization / Privilege**	**Major**	Resolved
MTC-15	The Logic Issue of `whitelist`	Logical Issue	Informational	Acknowledged
TCK-01	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved
TTC-01	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved
TTC-02	Delegation Not Moved Along With `transfer()` and `transferFrom()`	Logical Issue	Major	Resolved
TTC-03	Delegation Not Moved Along With `burn()`	Logical Issue	Major	Resolved
TTC-04	Privileged Ownership	**Centralization / Privilege**	**Minor**	Acknowledged
TTC-05	Centralized Risk In `safeTransfer()`	**Centralization / Privilege**	**Major**	Resolved
TTC-06	Centralized Risk In `safeTransferFrom()`	**Centralization / Privilege**	**Major**	Resolved
TTC-07	The Logic Issue of `whitelist`	Logical Issue	Informational	Acknowledged
TTC-08	Not Transfer Tokens in `transfer()` and `transferFrom()`	Logical Issue	Major	Acknowledged
WBN-01	Unlocked Compiler Version & Version Inconsistency	Language Specific	Informational	Resolved
WBN-02	Set `constant` to Variables	Gas Optimization	Informational	Resolved
WBN-03	Proper Usage of `public` and `external`	Gas Optimization	Informational	Resolved

AAL-01 | Unlocked Compiler Version & Version Inconsistency

Category	Severity	Location	Status
Language Specific	Informational	libs/AddrArrayLib.sol (moneytimefinance): 9	Resolved

Description

The contract contains unlocked compiler versions. An unlocked compiler version in the contract's source that code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be difficult to identify over a span of multiple compiler versions rather than a specific one. Examples:

- `pragma solidity 0.6.12;` `AddrArrayLib.sol#9`
- `pragma solidity >=0.5.0;` `Multicall.sol#3`
- `pragma solidity >0.4.18;` `WBNB.sol#3`
- `pragma solidity >=0.4.0;` `IBEP20Burnable.sol#3`

Recommendation

It is a general practice to alternatively lock the compiler at a specific version rather than allow a range of compiler versions to be utilized thus avoid compiler-specific bugs and be able to identify emerging ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

The client heeded our advice and resolved this issue by removing the `Multicall`,`IBEP20Burnable` and `WBNB` files.

IBE-01 | Unlocked Compiler Version & Version Inconsistency

Category	Severity	Location	Status
Language Specific	Informational	interface/IBEP20Burnable.sol (moneytimefinance): 3	Resolved

Description

The contract contains unlocked compiler versions. An unlocked compiler version in the contract's source that code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be difficult to identify over a span of multiple compiler versions rather than a specific one. Examples:

- `pragma solidity 0.6.12;` `AddrArrayLib.sol#9`
- `pragma solidity >=0.5.0;` `Multicall.sol#3`
- `pragma solidity >0.4.18;` `WBNB.sol#3`
- `pragma solidity >=0.4.0;` `IBEP20Burnable.sol#3`

Recommendation

It is a general practice to alternatively lock the compiler at a specific version rather than allow a range of compiler versions to be utilized thus avoid compiler-specific bugs and be able to identify emerging ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

The client heeded our advice and resolved this issue by removing the `Multicall`,`IBEP20Burnable` and `WBNB` files.

MCK-01 | Unlocked Compiler Version & Version Inconsistency

Category	Severity	Location	Status
Language Specific	Informational	libs/Multicall.sol (moneytimefinance): 3	Resolved

Description

The contract contains unlocked compiler versions. An unlocked compiler version in the contract's source that code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be difficult to identify over a span of multiple compiler versions rather than a specific one. Examples:

- `pragma solidity 0.6.12;` `AddrArrayLib.sol#9`
- `pragma solidity >=0.5.0;` `Multicall.sol#3`
- `pragma solidity >0.4.18;` `WBNB.sol#3`
- `pragma solidity >=0.4.0;` `IBEP20Burnable.sol#3`

Recommendation

It is a general practice to alternatively lock the compiler at a specific version rather than allow a range of compiler versions to be utilized thus avoid compiler-specific bugs and be able to identify emerging ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

The client heeded our advice and resolved this issue by removing the `Multicall`,`IBEP20Burnable` and `WBNB` files.

MCK-02 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	libs/Multicall.sol (moneytimefinance): 16~46	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

MCM-01 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefMoney.sol (moneytimefinance): 127, 135, 142, 152, 192, 339, 408, 489, 543, 552, 558	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

MCM-02 | Set `constant` to Variables

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefMoney.sol (moneytimefinance): 66, 93	Resolved

Description

The variables `maxShare` and `BURN_ADDRESS` are unchanged throughout the contract.

Recommendation

We advise the client to set `maxShare` and `BURN_ADDRESS` as `constant` variables.

Alleviation

The client heeded our advice and resolved this issue.

MCM-03 | Missing `indexed` in Events

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefMoney.sol (moneytimefinance): 99, 102	Resolved

Description

It is recommended to add the `indexed` keyword for parameters in events, which makes it easier for users to navigate event logs.

Recommendation

We advise the client to add keyword `indexed` in the declaration of events.

Alleviation

The client heeded our advice and resolved this issue.

MCM-04 | Lack of Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	MasterChefMoney.sol (moneytimefinance): 192, 256, 306, 344, 408, 489	Resolved

Description

There is no sanity check to validate if a pool exists.

Recommendation

We advise the client to adopt following modifier `validatePool` to functions `set()`,`pendingReward()`, `updatePool()`,`depositFor()`, `withdraw()` and `emergencyWithdraw()` as follows.

```
modifier validatePoolByPid(uint256 _pid) {
    require (_pid < poolInfo.length , "Pool does not exist") ;
    _;
}
```

Alleviation

The client heeded our advice and resolved this issue.

MCM-05 | Set `immutable` to Variables

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefMoney.sol (moneytimefinance): 60, 62, 87	Resolved

Description

The variables `money`, `busdToken` and `startBlock` are only changed once in the `constructor()` function.

Recommendation

We advise the client to set `money`, `busdToken` and `startBlock` as `immutable` variables.

Alleviation

The client heeded our advice and resolved this issue.

MCM-06 | Comparison to A Boolean Constant

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefMoney.sol (moneytimefinance): 571, 573	Resolved

Description

Comparison to a boolean constant.

```
571    if(found==false && amount > 0 ){ // add user
572       addr.pushAddress(_user);
573    }else if(found==true && amount == 0 ){ // remove user
574       addr.removeAddress(_user);
575    }
```

Recommendation

We advise the client to remove the comparison to the boolean constant like as follows

```
571    if(!found && amount > 0 ){ // add user
572       addr.pushAddress(_user);
573    }else if(found && amount == 0 ){ // remove user
574       addr.removeAddress(_user);
575    }
```

Alleviation

The client heeded our advice and resolved this issue.

MCM-07 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefMoney.sol (moneytimefinance): 543, 142	Resolved

Description

Functions that affect the status of sensitive variables should be able to emit events as notifications to customers.

- `updateMultiplier()`
- `dev()`
- `setAuthorizedCaller()`

Recommendation

We advise the client to add events for sensitive actions and emit them in the function as follows.

```
event MultiplierUpdated(uint256 multiplier);
event DevUpdated(address indexed oldDev, address indexed newDev);

function updateMultiplier(uint256 multiplierNumber) public onlyOwner {
    emit MultiplierUpdated(multiplierNumber);
    BONUS_MULTIPLIER = multiplierNumber;
}

function dev(address _devaddr) public {
    require(msg.sender == devaddr, "dev: wut?");
    emit DevUpdated(devaddr, _devaddr);
    devaddr = _devaddr;
}
```

Alleviation

The client heeded our advice and resolved this issue.

MCM-08 | `add()` Function Not Restricted

Category	Severity	Location	Status
Volatile Code	Major	MasterChefMoney.sol (moneytimefinance): 152	Acknowledged

Description

The comment in line L151, mentioned // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.

The total amount of reward `moneyReward` in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code does not reflect the comment behaviors as there is not any valid restriction on preventing this issue.

The current implementation is relying on the credibility of the owner to avoid redundantly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

We advise the client to ascertain whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate.
To achieve this, for example, we advise using mapping of `addresses` -> `booleans`, which can restrict the same address being added twice.

Alleviation

The client replies that they did not added double L protection as they need it to add 6 time pools.They don't use `balanceOf`, but store balance in a map of `pid`.

MCM-09 | Over Minted Token

Category	Severity	Location	Status
Logical Issue	Minor	MasterChefMoney.sol (moneytimefinance): 321~322	Acknowledged

Description

`updatePool()` function minted 100% + 8% (dev fee) of moneyReward.

Recommendation

We advise the client to mint 100% of the moneyReward instead of 100% + 8%.

Alleviation

The client replies that they prefer to keep 108% emission model.

MCM-10 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MasterChefMoney.sol (moneytimefinance): 20	Acknowledged

Description

The owner of the contract `MasterChefMoney` has the permission to:

- Update money reward count per block through `updateMoneyPerBlock()`
- Update `busd` reward count per block through `updateBusdPerBlock()`
- Update bonus multiplier through `updateMultiplier()`
- Transfer tokens to the current contract from any address through `updateEmissionSettings()`
- Update dev address through `dev()`
- Set tax address through `adminSetTaxAddr()`
- Set the rate of the tax through `adminSetTax()`
- Set white list through `adminSetWhiteList()`
- Enable `swapAndLiquifyEnabled`
- Update the given pool's time allocation point, lock period, deposit burn, and secondary reward through `set()`
- Update burn rate and emergency burn rate, which are the fractions the administrator can transfer to `BURN_ADDRESS` during withdraw.
- Update deposit fee, which is the fraction the administrator can transfer to the `BURN_ADDRESS`/`devaddr` during deposit
- Set burn rate to up to maxShare, lp tokens would be transferred to `BURN_ADDRESS`.
- Set authorized caller through `setAuthorizedCaller()`
- Set `busdFeeder1` through `SetBusdFeeder1()`
- Set `busdFeeder2` through `SetBusdFeeder2()`

without obtaining the consensus of the community.

Recommendation

We advise the client to renounce ownership and let the community govern when the timing is right with respect to transparency considerations.

Alleviation

The client replies that they will move the contract to `timelock` after deployment.

MCM-11 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	MasterChefMoney.sol (moneytimefinance): 72	Partially Resolved

Description

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

- Constants should be in UPPER_CASE_WITH_UNDERSCORES

In case the naming conventions are not followed, there should be proper documentation to explain the naming and the purpose of the variable.

Recommendation

We advise the client to use UPPER_CASE_WITH_UNDERSCORES for the aforementioned variables. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

The client heeded our advice and partially resolved this issue. For example: The `transferInsufficient` and `whitelistedTransfer` in the `TimeToken` and `MoneyToken` files. The `timePerBlockUpdated`,`depositRecipientUpdated` and `withdrawRecipientUpdated` in the `MasterChefTime` file.

MCM-12 | The Logical Issue of `depositFor()`

Category	Severity	Location	Status
Logical Issue	Critical	MasterChefMoney.sol (moneytimefinance): 344	Resolved

Description

When a user deposits into a particular pool, his/her pending reward from that pool is first transferred to the message sender's account on line 358. Due to the publicity of `depositFor()` function, another user may call it to transfer the `recipient`'s reward to his own `Money` account. This presents a vulnerability from malicious attempts to steal other's harvest.

Recommendation

We advise the client to change the `public` to the `internal`.

Alleviation

The client resolved this issue by modifying the logic.

MCM-13 | Calculation of `totalBusdAllocPoint`

Category	Severity	Location	Status
Logical Issue	Major	MasterChefMoney.sol (moneytimefinance): 221~223	Resolved

Description

Whether or not the pool contributes to the `totalBusdAllocPoint` depends on the flag `_secondaryReward` of the pool rather than the `allocPoint`'s changes. Thus, the flag `_secondaryReward` of the pool should be taken into account when updating the `totalBusdAllocPoint` in the function `set()`.

```
221     if(_secondaryReward) {
222         totalBusdAllocPoint =
totalBusdAllocPoint.sub(prevAllocPoint).add(_allocPoint);
223     }
```

Recommendation

We advise the client to update the `totalBusdAllocPoint` from the preceding `_secondaryReward` value instead of the current and not depends on the `allocPoint`.

Alleviation

The client heeded our advice and resolved this issue.

MCM-14 | Calling Function `userIndex` Before Balance Updating

Category	Severity	Location	Status
Control Flow	Medium	MasterChefMoney.sol (moneytimefinance): 496	Resolved

Description

There is a call to function `userIndex` before updating balance of user.

Recommendation

We advise the client to call the function `userIndex` after updating balance of user.

Alleviation

The client heeded our advice and resolved this issue.

MCM-15 | Calculation of `busdPendingReward`

Category	Severity	Location	Status
Logical Issue	Major	MasterChefMoney.sol (moneytimefinance): 289	Resolved

Description

The calculation of `busdPendingReward` depends on the `accBusdPerShare` rather than the constraints. The `busdPendingReward` is equal to Zero if the constraint does not hold. Thus, the calculation of `busdPendingReward` should be out of the constraints.

Recommendation

We advise the client to update the `busdPendingReward` in the same way as the `moneyPendingReward`.

Alleviation

The client heeded our advice and resolved this issue.

MCM-16 | `user.busdRewardDebt` Not Updated

Category	Severity	Location	Status
Logical Issue	Major	MasterChefMoney.sol (moneytimefinance): 484	Resolved

Description

It is intended that the `user.busdRewardDebt` be updated alongside corresponding transfers to keep track of the rewards already earned. Otherwise, the user would receive incorrect `busd` rewards as is the case of this issue.

Recommendation

We advise the client `user.busdRewardDebt` be updated alongside corresponding transfers.

Alleviation

The client heeded our advice and resolved this issue.

MCM-17 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	MasterChefMoney.sol (moneytimefinance): 543	Resolved

Description

The assigned value to `devaddr` should be verified as non-zero values to prevent being mistakenly assigned as `address(0)` in the `dev()` function.

Recommendation

We advise the client to check that the address is not zero in `dev()` like as follows.

```
require(_devaddr != address(0), "Zero address");
```

Alleviation

The client heeded our advice and resolved this issue.

MCM-18 | The Logic Issue of `add()`

Category	Severity	Location	Status
Logical Issue	Informational	MasterChefMoney.sol (moneytimefinance): 152	Resolved

Description

According to the logic of the `getBusdBalance()` function, the first pool should always use `busd` tokens. This restriction needs to be considered and preferably implemented in the `add()` function.

Recommendation

We advise the client to consider adding a restriction in the function `add()`.

Alleviation

The client heeded our advice and resolved this issue.

MCM-19 | The logical Issue of `emergencyWithdraw ()`

Category	Severity	Location	Status
Logical Issue	Minor	MasterChefMoney.sol (moneytimefinance): 492	Acknowledged

Description

If the `pool.lockPeriod` is set to positive, the users would not be able to withdraw. This contradicts the purpose of this function as a safe measure under emergency.

Alleviation

The client replies that there is no emergencyWithdraw feature in lock pools.

MCM-20 | Token Transfer In `updateEmissionSettings`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MasterChefMoney.sol (moneytimefinance): 534	Resolved

Description

The liquidity token for the first pool is intended to be `busd`. In this block of code, the `Owner` can simply pass `busd` of `_from` user to the current contract without the user's awareness let alone consent.

Recommendation

We would like to enquire about precautions against the `_from` is not the user address.

Alleviation

The client heeded our advice and resolved this issue by removing the input variable `_from`.

MCM-21 | The Logic Issue Of UnDistributed Rewards

Category	Severity	Location	Status
Logical Issue	Informational	MasterChefMoney.sol (moneytimefinance): 348	Acknowledged

Description

If `pool.lockPeriod` is set to positive in `depositFor()`, users would not be able to receive their pending `money` or `busd` reward and reward debts would not update.

Recommendation

We advise the client to review the function logic and would like to ascertain that it aligns with the design of your desire.

Alleviation

The client replies that there is no harvest action in lock pool, once user deposit again in lock pool. They save user's reward amount, that's why rewardDebt is not updated in lock pool.

MCM-22 | Set The `secondaryReward`

Category	Severity	Location	Status
Logical Issue	Informational	MasterChefMoney.sol (moneytimefinance): 56	Acknowledged

Description

It is not clarified how `busd` reward is calculated when `secondaryReward` is set to false. Multiple changes to `secondaryReward` across blocks could lead to incorrect pending rewards as the wrong calculation of rewards may be applied on these blocks. We would like to inquire about possible precautions. For example, suppose a pool is initialized with `SecondaryReward` set to `true`. After a few `Deposit` transactions, `AccuBushPershare` is updated and accumulates. The `owner` may then set `secondaryReward` and `_withUpdate` to `false` which causes `accBusd` to stop updating. `accBusdPerShare` will be accumulated to when the last deposit happen rather than to the time of setting. Some earlier depositers may find their claim to reward compromised due to such mechanism.

Alleviation

The client replies that `secondaryReward` will never be modified to be set to false. If they need to stop reward, they will set `busd` perBlock to 0.

Additionaly, `busd` reward distribution is a plus to users, users don't need to do any extra deposit to be rewarded in `busd`. So, they consider that there is no risk of user funds or reward loss.

MCM-23 | Update of the `rewardDebt` and `busdRewardDebt` in `depositFor()`

Category	Severity	Location	Status
Logical Issue	Major	MasterChefMoney.sol: 451	Resolved

Description

The `rewardDebt` and `busdRewardDebt` should be updated according to the latest 'amount'. Otherwise, the upcoming pending rewards would be larger than they should be, which could be exploited.

Recommendation

We advise the client to update the `rewardDebt` and `busdRewardDebt` after the `amount` modification.

```
user.rewardDebt = user.amount.mul(pool.accMoneyPerShare).div(1e12);
user.busdRewardDebt = user.amount.mul(pool.accBusdPerShare).div(1e12);
```

Alleviation

The client heeded our advice and resolved this issue.

MCM-24 | The logic of `payOrLockupPendingMoney()`

Category	Severity	Location	Status
Logical Issue	Major	MasterChefMoney.sol: 437~438	Resolved

Description

According to the current logic, the 'pending' and 'pendingBusd' are already pending rewards with lock-in rewards added. The `moneyRewardLockedUp` and `busdRewardLockedUp` are no need to add it up again respectively.

Recommendation

We advise the client to adopt as follows:

```
430     function payOrLockupPendingMoney(address _recipient, uint256 _pid) internal {
431         ......
432         user.moneyRewardLockedUp = pending;
433         user.busdRewardLockedUp = pendingBusd;
434         ......
435     }
```

Alleviation

The client heeded our advice and resolved this issue.

MCT-01 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefTime.sol (moneytimefinance): 107, 114, 119, 126, 139, 157, 218, 251, 278	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

MCT-02 | Set `immutable` to Variables

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefTime.sol (moneytimefinance): 55, 74	Resolved

Description

The variables `time` and `startBlock` are only changed once in the `constructor()` function.

Recommendation

We advise the client to set `time` and `startBlock` as `immutable` variables.

Alleviation

The client heeded our advice and resolved this issue.

MCT-03 | Set `constant` to Variables

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefTime.sol (moneytimefinance): 61, 76	Resolved

Description

The variables `maxShare` and `BURN_ADDRESS` are unchanged throughout the contract.

Recommendation

We advise the client to set `maxShare` and `BURN_ADDRESS` as `constant` variables.

Alleviation

The client heeded our advice and resolved this issue.

MCT-04 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefTime.sol (moneytimefinance): 114	Resolved

Description

Functions that affect the status of sensitive variables should be able to emit events as notifications to customers.

- `updateMultiplier()`

Recommendation

We advise the client to add events for sensitive actions and emit them in the function as follows.

```
event MultiplierUpdated(uint256 multiplier);

function updateMultiplier(uint256 multiplierNumber) public onlyOwner {
    emit MultiplierUpdated(multiplierNumber);
    BONUS_MULTIPLIER = multiplierNumber;
}
```

Alleviation

The client heeded our advice and resolved this issue.

MCT-05 | Missing `indexed` in Events

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefTime.sol (moneytimefinance): 83, 84, 85	Resolved

Description

It is recommended to add the `indexed` keyword for parameters in events, which makes it easier for users to navigate event logs.

Recommendation

We advise the client to add keyword `indexed` in the declaration of events.

Alleviation

The client heeded our advice and resolved this issue.

MCT-06 | `add()` Function Not Restricted

Category	Severity	Location	Status
Volatile Code	Major	MasterChefTime.sol (moneytimefinance): 139	Resolved

Description

The comment in line L138, mentioned // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.

The total amount of reward `timeReward` in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code does not reflect the comment behaviors as there is not any valid restriction on preventing this issue.

The current implementation is relying on the credibility of the owner to avoid redundantly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

We advise the client to ascertain whether the given pool for addition is a duplicate of an existing pool so that the pool addition is only successful when there is no duplicate.
To achieve this, for example, we advise using mapping of `addresses` -> `booleans`, which can restrict the same address being added twice.

Alleviation

The client heeded our advice and resolved this issue.

MCT-07 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Minor	MasterChefTime.sol (moneytimefinance): 218, 251, 278	Resolved

Description

The sequence of external call/transfer and storage manipulation must follow a check effect interaction pattern. For example

- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`

Recommendation

We advise the client to adopt `nonReentrant` modifier from `openzeppelin` library to the function `deposit()`,`withdraw()` and `emergencyWithdraw()` to prevent any reentrancy issue.(LINK)

Alleviation

The client heeded our advice and resolved this issue.

MCT-08 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Major	MasterChefTime.sol: 218, 251	Resolved

Description

The contract operates as the main entry for interaction with staking users. The staking users deposit LP tokens into the pool and in return get a proportionate share of the pool's rewards. Later on, the staking users can withdraw their own assets from the pool. In this procedure, `deposit()` and `withdraw()` are involved in transferring users' assets into (or out of) the protocol. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged (and burned) transaction fee. As a result, this may not meet the assumption behind these low-level asset-transferring routines and will bring unexpected balance inconsistencies.

Recommendation

We advise the client to regulate the set of LP tokens supported in MoneyTime. If there is a need to support deflationary tokens, add necessary mitigation mechanisms to keep track of accurate balances.

Alleviation

The client heeded our advice and resolved this issue.

MCT-09 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MasterChefTime.sol (moneytimefinance): 22	Acknowledged

Description

The owner of the contract `MasterChefTime` has the permission to:

- Update reward count per block through `updateTimePerBlock()`
- Update bonus multiplier through `updateMultiplier()`
- Update the address of deposit recipient through `updateDepositRecipient()`
- Update the address of the withdrawing recipient through `updateWithdrawRecipient()`
- Update the given pool's time allocation point through `set()`
- Add a new pool through `add()`
- Update withdrawal fee which is the fractions the administrator can transfer to `withdrawRecipient` during withdraw.
- Update deposit fee, which is the fraction the administrator can transfer to the `BURN_ADDRESS`/`depositRecipient` during deposit

without obtaining the consensus of the community.

Recommendation

We advise the client to renounce ownership and let the community govern when the timing is right with respect to transparency considerations.

Alleviation

The client replies that they will move the contract to `timelock` after deployment.

MCT-10 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	MasterChefTime.sol (moneytimefinance): 65, 82, 83, 84	Partially Resolved

Description

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

- Constants should be in UPPER_CASE_WITH_UNDERSCORES

In case the naming conventions are not followed, there should be proper documentation to explain the naming and the purpose of the variable.

Recommendation

We advise the client to use UPPER_CASE_WITH_UNDERSCORES for the aforementioned variables. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

The client heeded our advice and partially resolved this issue. For example: The `transferInsufficient` and `whitelistedTransfer` in the `TimeToken` and `MoneyToken` files. The `timePerBlockUpdated`,`depositRecipientUpdated` and `withdrawRecipientUpdated` in the `MasterChefTime` file.

MCT-11 | Lack of Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	MasterChefTime.sol (moneytimefinance): 157, 177, 200, 218, 251, 278	Resolved

Description

There's no sanity check to validate if a pool exists.

Recommendation

We advise the client to adopt following modifier `validatePool` to functions `set()`, `pendingTime()`, `updatePool()`, `deposit()`, `withdraw()` and `emergencyWithdraw()` as follows.

```
modifier validatePoolByPid(uint256 _pid) {
    require (_pid < poolInfo.length , "Pool does not exist") ;
    _;
}
```

Alleviation

The client heeded our advice and resolved this issue.

MCT-12 | Comparison to A Boolean Constant

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChefTime.sol: 411, 413	Resolved

Description

Comparison to a boolean constant.

```
411    if(found==false && amount > 0 ){ // add user
412       addr.pushAddress(_user);
413    }else if(found==true && amount == 0 ){ // remove user
414       addr.removeAddress(_user);
415    }
```

Recommendation

We advise the client to remove the comparison to the boolean constant like as follows

```
408    AddrArrayLib.Addresses storage addr = addressByPid[_pid];
409
410    uint256 amount = userInfo[_pid][_user].amount;
411    if( amount > 0 ){ // add user
412        addr.pushAddress(_user);
413    }else if( amount == 0 ){ // remove user
414        addr.removeAddress(_user);
415    }
```

Alleviation

The client heeded our advice and resolved this issue.

MCT-13 | Over Minted Token

Category	Severity	Location	Status
Logical Issue	Minor	MasterChefTime.sol: 267	Acknowledged

Description

`updatePool()` function minted 100% + 10% (dev fee) of timeReward.

Recommendation

We advise the client to mint 100% of the timeReward instead of 100% + 10%.

Alleviation

The client replies that they prefer to keep 110% emission model.

MTC-01 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	MoneyToken.sol (moneytimefinance): 255, 412, 266	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

MTC-02 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	MoneyToken.sol (moneytimefinance): 250, 271, 255	Resolved

Description

The input variables `_taxToAddrAddress`, `router` should be verified as non-zero values to prevent being mistakenly assigned as `address(0)` in the `constructor()`, `setTaxAddr()` and `init_router()` functions respectively.

Recommendation

We advise the client to check that the addresses are not zero by adding the following check-in the `constructor()`, `setTaxAddr()` and `init_router()` like as follows.

```
require(address(_taxToAddrAddress) != address(0), "Zero address");
```

Alleviation

The client heeded our advice and resolved this issue.

MTC-03 | Delegation Not Moved Along With `transfer()` and `transferFrom()`

Category	Severity	Location	Status
Logical Issue	Major	MoneyToken.sol (moneytimefinance): 287, 290, 303, 305, 313, 316, 329, 331	Resolved

Description

The voting power of delegation is not moved from token sender to token recipient along with the `transfer()` and `transferFrom()`. Current `transfer()` and `transferFrom()` are from `BEP20` protocol and doesn't invoke `_moveDelegates()`.

Recommendation

We advise the client to invoke `_moveDelegates()` while re-implementing `transfer()` and `transferFrom()`. This would ensure the proper transfer of power along with tokens.

Alleviation

The client heeded our advice and resolved this issue.

MTC-04 | Typos In The Contract

Category	Severity	Location	Status
Coding Style	Informational	MoneyToken.sol (moneytimefinance): 240, 392	Resolved

Description

There are several typos in the code and comments.

1. In the following code snippet, `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

```
237 event SwapAndLiquify(
238   uint256 tokensSwapped,
239   uint256 ethReceived,
240   uint256 tokensIntoLiqudity
241 );
```

2. `recieve` should be `receive` and `swaping` should be `swapping` in the line of comment `//to recieve ETH from uniswapV2Router when swaping`.

Recommendation

We recommend correcting all typos in the contract.

Alleviation

The client heeded our advice and resolved this issue.

MTC-05 | Contract Gains Non-withdrawable ETH Via The `swapAndLiquify` Function

Category	Severity	Location	Status
Logical Issue	Major	MoneyToken.sol (moneytimefinance): 336	Resolved

Description

The `swapAndLiquify` function converts half of the `contractTokenBalance` Money tokens to ETH. The other half of Money tokens and part of the converted ETH are deposited into the Money-ETH pool on uniswap as liquidity. For every `swapAndLiquify` function call, a small amount of ETH leftover in the contract. This is because the price of Money drops after swapping the first half of Money tokens into ETHs, and the other half of Money tokens require less than the converted ETH to be paired with it when adding liquidity. The contract does not appear to provide a way to withdraw those ETH, and they will be locked in the contract forever.

Recommendation

It is not ideal that more and more ETH are locked into the contract over time. The simplest solution is to add a `withdraw` function in the contract to withdraw ETH. Other approaches that benefit the Money token holders can be:

- Distribute ETH to Money token holders proportional to the amount of token they hold.
- Use leftover ETH to buy back Money tokens from the market to increase the price of Money.

Alleviation

The client heeded our advice and resolved this issue by adding the function `claimBNB()`.

MTC-06 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	Informational	MoneyToken.sol (moneytimefinance): 382	Resolved

Description

The return values of function `addLiquidityETH` are not properly handled.

```
382     uniswapV2Router.addLiquidityETH{value: ethAmount}(
383         address(this),
384         tokenAmount,
385         0, // slippage is unavoidable
386         0, // slippage is unavoidable
387         owner(),
388         block.timestamp
389     );
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

Alleviation

The client heeded our advice and resolved this issue.

MTC-07 | Centralized Risk In `addLiquidity`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MoneyToken.sol (moneytimefinance): 387	Resolved

Description

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function with the `to` address specified as `owner()` for acquiring the generated LP tokens from the `Money-ETH` pool. As a result, over time the `_owner` address will accumulate a significant portion of LP tokens. If the `_owner` is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation

We advise the `to` address of the `uniswapV2Router.addLiquidityETH` function call to be replaced by the contract itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

The client heeded our advice and resolved this issue.

MTC-08 | Third Party Dependencies

Category	Severity	Location	Status
Control Flow	Minor	MoneyToken.sol (moneytimefinance): 232	Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party UniSwap protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties may be compromised which leads to assets being lost or stolen.

Recommendation

We understand that the business logic of the Money protocol requires the interaction UniSwap protocol for adding liquidity to the Money-ETH pool and swap tokens. We encourage the team to constantly monitor the statuses of those 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

The client replies that the UniSwap contracts (VANI) are deployed by them.

MTC-09 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	MoneyToken.sol (moneytimefinance): 271, 275, 280	Resolved

Description

Functions that affect the status of sensitive variables should be able to emit events as notifications to customers.

- `setTaxAddr()`
- `setTax()`
- `setWhiteList()`

Recommendation

We advise the client to add events for sensitive actions and emit them in the function.

Alleviation

The client heeded our advice and resolved this issue.

MTC-10 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	MoneyToken.sol (moneytimefinance): 223	Acknowledged

Description

The owner of the contract `MoneyToken` has the permission to:

- Mint uncapped tokens to any address through `mint()`
- Set tax address through `setTaxAddr()`
- Set the rate of the tax through `setTax()`
- Set white list through `setWhiteList()`
- Enable `swapAndLiquifyEnabled`
- Init router through `init_router()`

without obtaining the consensus of the community.

Recommendation

We advise the client to renounce ownership and let the community govern when the timing is right with respect to transparency considerations.

Alleviation

The client replies that they will move the contract to `timelock` after deployment.

MTC-11 | Comparison to A Boolean Constant

Category	Severity	Location	Status
Gas Optimization	Informational	MoneyToken.sol (moneytimefinance): 289, 315	Resolved

Description

Comparison to a boolean constant.

```
289  if( whitelist[_to] == true ){
290     transfer(_to, _total);
291  }
```

```
315  if( whitelist[_to] == true ){
316     transferFrom(_sender, _to, _total);
317  }
```

Recommendation

We advise the client to remove the comparison to the boolean constant like as follows

```
289  if( whitelist[_to] ){
290     transfer(_to, _total);
291  }
```

```
315  if( whitelist[_to] ){
316     transferFrom(_sender, _to, _total);
317  }
```

Alleviation

The client heeded our advice and resolved this issue.

MTC-12 | Centralized Risk In `safeTransfer()`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MoneyToken.sol (moneytimefinance): 284	Resolved

Description

Owner has the privilege to transfer tokens on the current contract to any designated address. The balance value should be msg.sender's balance not the contract‘s.

Alleviation

The client heeded our advice and resolved this issue by removing the function.

MTC-13 | Centralized Risk In `safeTransferFrom()`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MoneyToken.sol (moneytimefinance): 310	Resolved

Description

Owner has the privilege to transfer tokens from any authorized `sender` to any designated address.

Alleviation

The client heeded our advice and resolved this issue by removing the function.

MTC-14 | Centralized Risk In `claimBNB()`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	MoneyToken.sol: 320	Resolved

Description

The `taxToAddrAddress` has the privilege to transfer tokens from the contract.

Alleviation

The client heeded our advice and resolved this issue by removing `claimBNB()` and sending dust directly to `taxToAddrAddress` in the `addLiquidity()` function.

MTC-15 | The Logic Issue of `whitelist`

Category	Severity	Location	Status
Logical Issue	Informational	MoneyToken.sol: 181, 232	Acknowledged

Description

According to current whitelist logic, the tokens could be only be transferred as long as one party' is on the white list. However even if neither parties are on the whitelist, they can still achieve mutual transfer of tokens by routing through a third-party whitelist address. We wonder if this suits your design.

Alleviation

The client replies that they know this conditions, they clarify that only their contracts and management would be whitelisted to do `TIME` token transfers according to their token economics.

TCK-01 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	Timelock.sol (moneytimefinance): 54, 63, 71, 84, 95, 104	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

TTC-01 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	TimeToken.sol (moneytimefinance): 11	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

TTC-02 | Delegation Not Moved Along With `transfer()` and `transferFrom()`

Category	Severity	Location	Status
Logical Issue	Major	TimeToken.sol (moneytimefinance): 9	Resolved

Description

The voting power of delegation is not moved from token sender to token recipient along with the `transfer()` and `transferFrom()`. Current `transfer()` and `transferFrom()` are from `BEP20` protocol and doesn't invoke `_moveDelegates()`.

Recommendation

We advise the client to invoke `_moveDelegates()` while re-implementing `transfer()` and `transferFrom()`. This would ensure the proper transfer of power along with tokens.

Alleviation

The client heeded our advice and resolved this issue.

TTC-03 | Delegation Not Moved Along With `burn()`

Category	Severity	Location	Status
Logical Issue	Major	TimeToken.sol (moneytimefinance): 16	Resolved

Description

Whenever new Time tokens are minted, new delegates are moved from the zero address to the recipient of the minting process. However, whenever tokens are burned, new delegates are moved from the recipient to the zero address.

Recommendation

We advise the client to invoke `_moveDelegates()` like as follows.

```
15    function burn(address _to ,uint256 _amount) external onlyOwner {
16         _burn(_to, _amount);
17         _moveDelegates(_delegates[_to], address(0), _amount);
18     }
```

Alleviation

The client heeded our advice and resolved this issue by removing the `burn()`.

TTC-04 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	TimeToken.sol (moneytimefinance): 11, 15	Acknowledged

Description

To bridge the trust gap between owner and users, the owner needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The owner has the responsibility to notify users with the following capability:

- Owner has the privilege to mint uncapped tokens through `mint()`
- Owner has the privilege to set white list through `setWhiteList()`

Recommendation

We advise the client to renounce ownership and let the community govern when the timing is right with respect to transparency considerations.

Alleviation

The client replies that token contracts will be owned by MasterChef contracts, MasterChef contracts will be owned by timelock.

TTC-05 | Centralized Risk In `safeTransfer()`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	TimeToken.sol: 136	Resolved

Description

Owner has the privilege to transfer tokens on the current contract to any designated address. The balance value should be msg.sender's balance not the contract‘s.

Alleviation

The client heeded our advice and resolved this issue by removing the function.

TTC-06 | Centralized Risk In `safeTransferFrom()`

Category	Severity	Location	Status
Centralization / Privilege	**Major**	TimeToken.sol: 146	Resolved

Description

Owner has the privilege to transfer tokens from any authorized `sender` to any designated address.

Alleviation

The client heeded our advice and resolved this issue by removing the function.

TTC-07 | The Logic Issue of `whitelist`

Category	Severity	Location	Status
Logical Issue	Informational	TimeToken.sol: 118, 157	Acknowledged

Description

According to current whitelist logic, the tokens could be only be transferred as long as one party' is on the white list. However even if neither parties are on the whitelist, they can still achieve mutual transfer of tokens by routing through a third-party whitelist address. We wonder if this suits your design.

Alleviation

The client replies that they know this conditions, they clarify that only their contracts and management would be whitelisted to do `TIME` token transfers according to their token economics.

TTC-08 | Not Transfer Tokens in `transfer()` and `transferFrom()`

Category	Severity	Location	Status
Logical Issue	Major	TimeToken.sol: 117, 156	Acknowledged

Description

According to current whitelist logic, the tokens could be only be transferred as long as one party is on the white list. If the parties are not on the whitelist, they could not transfer. This is not the ERC20 definition.

Alleviation

The client replies that in MoneyTime TokenEconomic, the `$TIME` token cannot be sold or traded. The only purpose of the `$TIME` token is to be “STAKED” in the TimePools to earn `$MONEY`. There is no $TIME liquidity. It's a tool token like `Syrup` for `pancakeswap`.

(LINK) This is specifically noted in the roadmap, communicated to the community, and display on the farming pool page where users earn `$TIME`.

WBN-01 | Unlocked Compiler Version & Version Inconsistency

Category	Severity	Location	Status
Language Specific	Informational	libs/WBNB.sol (moneytimefinance): 3	Resolved

Description

The contract contains unlocked compiler versions. An unlocked compiler version in the contract's source that code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be difficult to identify over a span of multiple compiler versions rather than a specific one. Examples:

- `pragma solidity 0.6.12;` `AddrArrayLib.sol#9`
- `pragma solidity >=0.5.0;` `Multicall.sol#3`
- `pragma solidity >0.4.18;` `WBNB.sol#3`
- `pragma solidity >=0.4.0;` `IBEP20Burnable.sol#3`

Recommendation

It is a general practice to alternatively lock the compiler at a specific version rather than allow a range of compiler versions to be utilized thus avoid compiler-specific bugs and be able to identify emerging ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

The client heeded our advice and resolved this issue by removing the `Multicall`,`IBEP20Burnable` and `WBNB` files.

WBN-02 | Set `constant` to Variables

Category	Severity	Location	Status
Gas Optimization	Informational	libs/WBNB.sol (moneytimefinance): 6~8	Resolved

Description

The variables `name`, `symbol` and `decimals` are unchanged throughout the contract.

Recommendation

We advise the client to set `name`, `symbol` and `decimals` as `constant` variables.

Alleviation

The client heeded our advice and resolved this issue by removing the `WBN` file.

WBN-03 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	Informational	libs/WBNB.sol (moneytimefinance): 18, 25, 32, 36, 42	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

The client heeded our advice and resolved this issue.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.