CERTIK

Audit Report

Produced by CertiK

for Multiplier Finance

Contents

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 Verification Services Agreement between CertiK and Multiplier Finance (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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 without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Multiplier Finance** to discover issues and vulnerabilities in the source code of their **ERC-20 Smart Contracts** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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.

Testing Summary

SECURITY LEVEL

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by Multiplier Finance.

This audit was conducted to discover issues and vulnerabilities in the source code of Multiplier Finance's ERC-20 Smart Contracts.

TYPE	Smart Contract
SOURCE CODE	https://github.com/Multiplier-Finance/SmartContracts/
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	Aug 13, 2020
DELIVERY DATE	Sept 1, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the Multiplier Finance team to audit the design and implementation of their ERC-20 smart contracts and its compliance with the EIPs it is meant to implement.

The audited source code link is:

- https://github.com/Multiplier-Finance/SmartContracts/commit/b3c00f99cd17d202e650c7c9d28c758e7715c59a

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

Documentation

The sources of truth regarding the operation of the contracts in scope were minimal although the token fulfilled a simple use case we were able to fully assimilate. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Multiplier Finance team or reported an issue.

Summary

The project's codebase is a typical EIP20 implementation with additional support for a yield contract creation mechanism.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, although **two minor and three medium flaws** were identified that should be remediated as soon as possible to ensure the contracts of the Multiplier Finance team are of the highest standard and quality.

The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and as such its typical ERC-20 functions **can be deemed to be of high security and quality**.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report **to achieve a high standard of code quality and security**.

Additionally, the code segments regarding **the medium findings should be revised,** in order to eliminate the possibility of a vulnerability exploitation.

Findings

Exhibit 1

TITLE	TYPE	SEVERITY	LOCATION
Contract Name	Coding Style	Informational	yieldContract.sol

[INFORMATIONAL] Description:

Within Solidity, it is a generally accepted convention to name contracts using CapWord case and have the Solidity file that represents them utilize the same name.

Recommendations:

We advise that the contract's name is set to *YieldContract.sol*.

Alleviations:

The team opted to take our recommendation into account and adhered to the standard Solidity naming conventions.

Exhibit 2

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version	Language Specific Issue	Informational	All “pragma” statements

[INFORMATIONAL] Description:

An unlocked compiler version in the source code of the contract 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 an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendations:

We advise that the compiler version is instead locked at the lowest version possible that the full project can be compiled at. This is evident across the codebase on multiple contracts and will only be mentioned once, ***however it applies to all.***

Alleviations:

The team opted to take our recommendation into account and locked the compiler version to v0.6.2 .

Exhibit 3

TITLE	TYPE	SEVERITY	LOCATION
Non-Standard Naming Convention	Coding Style	Informational	yieldContract: L76 - L79

[INFORMATIONAL] Description:

The linked `enums` do not conform to the Solidity style guide, whereby each state of the `enum` should be named in CapWords style.

Recommendations:

We advise that the corresponding states are adapted to utilize that convention.

Alleviations:

The team opted to take our recommendation into account and adhered to the Solidity naming conventions.

Exhibit 4

TITLE	TYPE	SEVERITY	LOCATION
Redundant Assignment	Optimization	Informational	yieldContract: L199

[INFORMATIONAL] Description:

Within Solidity, variables are initialized to their default value which in most cases is zero. As such, the contract-level assignment of zero here is ineffectual.

Recommendations:

We advise the removal of those assignments.

Alleviations:

The team opted to take our recommendation into account and removed the redundant code.

Exhibit 5

TITLE	TYPE	SEVERITY	LOCATION
Variable Mutability Specifiers	Coding Style	Informational	yieldContract: L127-L130, L132-L135, L200, L202-L203,

[INFORMATIONAL] Description:

The linked variables are only assigned to once during the constructor of the contract and as such can be declared as immutable if a compiler version of v0.6.5 and up is utilized.

Recommendations:

Otherwise, we advise that they are set to constant variables that are directly assigned in their contract declaration as both of those types significantly reduce the gas cost involved with interacting with them.

Alleviations:

The team opted to take our recommendation into account, assigned values to the `constant` variables in their contract declaration, but also used a compiler version lower than v0.6.5, i.e. v0.6.2 .

Exhibit 6

TITLE	TYPE	SEVERITY	LOCATION
Redundant `SafeMath`	Optimization	Informational	yieldContract: L192-L198, L518 (Partial), L529, L575

[INFORMATIONAL] Description:

The linked variables utilize literals to conduct SafeMath operations. As these statements are never going to trigger overflows / underflows, it is safe to simply omit the SafeMath utilization and conduct them in their raw format i.e. 50 * (10**4) instead of SafeMath.mul(50, 10 ** 4).

Additionally, as these variables are meant to represent percentages according to the preceding comments of the variables they are being assigned to, it is more legible to utilize decimal representation as it is evaluated properly by the compiler i.e. 0.05 * (10**6) instead of 50 * (10**4).

Recommendations:

We advise the removal of those invocations.

Alleviations:

The team opted to take our recommendation into account and removed the `SafeMath` operations.

Exhibit 7

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Utilization of `SafeMath`	Optimization	Informational	yieldContract: L453, L457, L517-L519, L522, L525, L528-L530, L536, L564-L566, L569-L571, L574-L576, L579, L600, L692, L696

[INFORMATIONAL] Description:

The codebase of the contract utilizes the SafeMath library implementation by referencing it directly and accessing the various methods it exposes i.e. SafeMath.sub(validERC20.length, 1). While this utilization will work, L38 overloads the uint256 primitive meaning those same methods are exposed directly by the variables.

Recommendations:

Thus, we advise all linked statements to be simplified to greatly aid in the legibility of the codebase.

As an example:

```
SafeMath.mul(
    SafeMath.mul(
        SafeMath.mul(
            SafeMath.mul(
                _collateral,
                ERC20Map[_ERC20Address].mFactor
            ),
            tenureApyMap[_tenure]
        ),
        10 ** uint256(ERC20(mxxAddress).decimals())),
    _tenure
)
```

Would be reduced to:

```
_collateral
    .mul(ERC20Map[_ERC20Address].mFactor)
    .mul(tenureApyMap[_tenure])
    .mul(10 ** uint256(ERC20(mxxAddress).decimals()))
    .mul(_tenure)
```

Alleviations:

The team opted to take our recommendation into account and simplified the `SafeMath` statements.

Exhibit 8

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Function Implementation	Volatile Code	Informational	yieldContract: L220-L234

[INFORMATIONAL] Description:

The `getIndex` function is meant to lookup for a specific address in an array of addresses and return its index if found, otherwise it will revert. The current implementation is not gas efficient for multiple reasons.

First, it utilizes `uint32` data types for both the return variable as well as the for loop iterator. The EVM Solidity executes in is optimized to handle full word data types, a.k.a. 32-byte / 256-bit, meaning that handling `uint32` data types causes it to utilize more gas for padding and unpadding operations. Secondly, the named return variable is assigned to and returned in the immediate statement that follows. This renders naming the return variable redundant.

Lastly, the programming paradigm imposed here is incorrect. It is ill-advised to utilize the revert function in Solidity code.

Recommendations:

We advise that a traditional `uint256` is utilized instead. Also, the named return variable can be omitted and the i returned directly. Lastly, a boolean could be utilized as a returned value instead of the revert function call, which indicates whether the element was found, leaving the caller to decide whether to require the boolean or continue execution.

Alleviations:

The team opted to take our recommendation into account and return the loop index with the optimal type, i.e. `uint256`, and a boolean value of whether the element was found or not.

Exhibit 9

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Key Type	Optimization	Informational	yieldContract: L119, L248-L251

[INFORMATIONAL] Description:

Within Solidity, `mapping` lookup operations conduct a `keccak256` cycle on the input variable and utilize it for assessing the offset in storage. This means that a `uint256` key data type would cost less to utilize than a `uint16` data type, as the latter would need to be unpadded and padded whenever utilized in the hashing operation involved in the mapping lookup.

Recommendations:

We advise that a `uint256` data type is used instead.

Alleviations:

The team opted to take our recommendation into account and used the optimal data type, namely `uint256`.

Exhibit 10

TITLE	TYPE	SEVERITY	LOCATION
Unsanitized Parameters	Volatile Code	Minor	yieldContract: L261-L271

[MINOR] Description:

The parameters of the protocol set by the `setParamType` function should be vetted to ensure that they comply with certain logical assumptions, such as that `minEarlyRedeemFee` must always be less-than-or-equal (<=) to `maxEarlyRedeemFee` and `totalAllocatedMxx` should be set to a value greater-than-or-equal (>=) to `mxxMintedFromContract`.

Recommendations:

We advise the team to add `require` statements for the specific cases.

Alleviations:

The team opted to take our recommendation into account and added the `require` statements, as described.

Exhibit 11

TITLE	TYPE	SEVERITY	LOCATION
Differentiation of Functionality Based On Literal	Optimization	Informational	yieldContract: L284-L306, L382-L391, L399-L407, L482-L543, L552-L603, L612-L635, L644-L675

[INFORMATIONAL] Description:

The `_ERC20Address` has a dual-purpose whereby it is also able to represent Ether, meaning that it may also represent a value that is not an ERC20 address.

Recommendations:

As such, we advise that any comparisons utilizing `address(0)` to differentiate functionality instead utilize a contract-level declared constant which is equivalent to the current code gas-wise.

Alleviations:

The team opted to take our recommendation into account and added a contract-level constant variable.

Exhibit 12

TITLE	TYPE	SEVERITY	LOCATION
`if` Clause to Ternary	Coding Style	Informational	yieldContract: L293-L303

[INFORMATIONAL] Description:

The linked code statement would benefit in simplicity by utilizing the ternary logical operator as only the first argument of the `ERC20Details` `struct` instantiation is affected.

Recommendations:

We advise the team to change this code block to ternary (condition ? statement : statement).

Alleviations:

The team opted to take our recommendation into account and used a ternary operation.

Exhibit 13

TITLE	TYPE	SEVERITY	LOCATION
Redundant `delete` Operation	Optimization	Informational	yieldContract: L351-L353

[INFORMATIONAL] Description:

The `pop` function of a dynamic array in Solidity automatically invokes delete on the element to-be-popped, as such the preceding statement of the `pop` operation is redundant.

Recommendations:

We advise the team to remove the redundant code.

Alleviations:

The team opted to take our recommendation into account and removed the redundant code.

Exhibit 14

TITLE	TYPE	SEVERITY	LOCATION
Delist / Undelist to Setter Function	Optimization	Informational	yieldContract: L360-L391

[INFORMATIONAL] Description:

The linked `delistValidERC20` and `undelistValidERC20` functions could instead utilize a single setter function that adjusts the list status of an ERC20 based on a boolean input variable greatly reducing the generated bytecode.

Recommendations:

We advise the team to implement the Setter function, as stated above.

Alleviations:

The team opted to take our recommendation into account and implemented the “Setter” function, as described.

Exhibit 15

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual `require` Check	Volatile Code	Informational	yieldContract: L385

[INFORMATIONAL] Description:

The logical require check imposed here is ineffectual as it simply ensures the address that is about to be listed is a contract or represents ETH. This does not necessarily mean it has been normally added as a `ValidERC20` meaning that the function is able to be executed on unadded ERC20s.

Recommendations:

We advise that a lookup on `ERC20Map` is done instead that ensures at least one of the resulting struct variables is non-zero.

Alleviations:

The team opted to take our recommendation into account and added the `require` statement, as described.

Exhibit 16

TITLE	TYPE	SEVERITY	LOCATION
`require` to `modifier`	Optimization	Informational	yieldContract: L287, L385, L402

[INFORMATIONAL] Description:

The linked `require` statements can instead be grouped to a single modifier as they are utilized numerous times across the codebase.

Recommendations:

We advise that implement a modifier, as stated above.

Alleviations:

The team opted to take our recommendation into account, implemented the `onlyERC20OrETH` modifier and introduced it to the code segments needed.

Exhibit 17

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Function Implementation	Volatile Code & Optimization	Informational	yieldContract: L439-L465, L685-L706

[INFORMATIONAL] Description:

Once again, the `uint32` data type is utilized across the function whilst `uint256` data types would be safer and cost less gas due to the absence of casting operations.

Additionally, the current implementation is incorrect in the sense that if `_start` is greater than `validERC20.length` and `_end` is greater than or equal to `_start`, the function will throw without a correct error message. To avoid this, the `require` statement of L449 should be relocated after the subsiding `if` clause.

The return variable could also be named to `subsetValidERC20` to avoid the last return statement.

We should note that this function, despite its comment stating that "valid ERC20 contracts" are being returned, would return contracts that have their `isValid` variable set to false.

Similar optimizations can be set to sub-exhibit B of this finding.

Recommendations:

We advise that implements the references of the “Description” section.

Alleviations:

The team opted to take our recommendation into account and changed the linked code, strictly adhering to our references.

Exhibit 18

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Data Type	Optimization	Informational	yieldContract: L105

[INFORMATIONAL] Description:

As the `struct` member `contractStatus` is meant to represent a `Status`, it is possible to declare it as such instead of casting it to a `uint8` wherever it is utilized.

Recommendations:

We advise that the team changes the type of the `struct` member `contractStatus` from a `uint8` to a `Status` `enum`.

Alleviations:

The team opted to take our recommendation into account and changed the data type from `uint8` to a `Status` `enum`.

Exhibit 19

TITLE	TYPE	SEVERITY	LOCATION
Redundant Use of `abi.encodePacked`	Optimization	Informational	yieldContract: L487-L488

[INFORMATIONAL] Description:

Whenever identifiers are desired, `abi.encode` should be favored over `abi.encodePacked` as the latter is prone to hash-collisions on different inputs due to its tight-packing mechanism. In this particular instance, hash-collision is impossible however `abi.encode` would cost less gas as no tight packing can be done on the input variables.

Recommendations:

We advise the use of `abi.encode` over `abi.encodePacked`.

Alleviations:

The team opted to take our recommendation into account and favored `abi.encode` over `abi.encodePacked`.

Exhibit 20

TITLE	TYPE	SEVERITY	LOCATION
Conditional Optimizations	Optimization	Informational	yieldContract: L499, L508

[INFORMATIONAL] Description:

The linked `require` conditionals both check the “same” condition separately in the same `if` clause. As such, the conditional “*require(_collateral != 0,"Collateral is 0")*” should precede the `if` block and the “*msg.value > 0*” conditional should be removed from L499.

Recommendations:

We advise that implements the references of the “Description” section.

Alleviations:

The team opted to take our recommendation into account and optimized the code block, as described.

Exhibit 21

TITLE	TYPE	SEVERITY	LOCATION
Overflow-Prone	Volatile Code & Optimization	Minor	yieldContract: L517-L518

[MINOR] Description:

The linked mathematical statements are highly prone to overflow if tokens with a high number of decimal precision are utilized to the consequent multiplications.

Recommendations:

To avoid such overflows, it is advised that the multiplications and divisions are coupled, losing as little precision as possible to rounding.

Alleviations:

The team opted to take our recommendation into account, utilizing the multiplications before the division, losing the minimal precision.

Exhibit 22

TITLE	TYPE	SEVERITY	LOCATION
Operation Duplication	Optimization	Informational	yieldContract: L521-L525

[INFORMATIONAL] Description:

As the evaluation of "*SafeMath.add(mxxMintedFromContract, valueToBeMinted)*" is utilized twice.

Recommendations:

We advise that the order of those two statements is reversed and the `require` statement instead reads the value of `mxxMintedFromContract` which after optimizations should reside in `memory`.

Alleviations:

The team opted to take our recommendation into account and reversed the two statements described, as well as used `memory` to store the optimizations.

Exhibit 23

TITLE	TYPE	SEVERITY	LOCATION
Operation Rounding	Volatile Code	Medium	yieldContract: L527-L530

[MEDIUM] Description:

The percentage operation conducted here can potentially yield zero for a non-zero amount of `valueToBeMinted`, meaning that it would be possible to bypass the burning fee with small segments of a larger value i.e. if I want to mint 1000000 tokens I would instead mint 100 batches of 10000 to avoid the fee.

Recommendations:

We advise the team revises the code segment.

Alleviations:

The team opted to take our recommendation into account and used `SafeMath` operations to calculate the `valueToBeMinted`, as a minimal precaution to this problem.

Exhibit 24

TITLE	TYPE	SEVERITY	LOCATION
Conditional Optimization	Optimization	Informational	yieldContract: L561

[INFORMATIONAL] Description:

The former part of the conditional can be skipped as the `startTime` of a yield contract is always set during its construction to `now`, meaning that `now` will always be greater-than-or-equal to `startTime`.

Recommendations:

We advise the team removes the conditional segment stated above.

Alleviations:

The team opted to take our recommendation into account and removed the former part of the conditional.

Exhibit 25

TITLE	TYPE	SEVERITY	LOCATION
Redundant Calculations	Optimization	Informational	yieldContract: L565, L570

[INFORMATIONAL] Description:

The statement of sub-Exhibit A (L565) and sub-Exhibit B (L570) are exactly the same, meaning that sub-Exhibit B can safely be omitted.

Recommendations:

We advise the team to remove the sub-Exhibit B stated above.

Alleviations:

The team opted to take our recommendation into account and removed the redundant code.

Exhibit 26

TITLE	TYPE	SEVERITY	LOCATION
Rounding Issue	Volatile Code	Medium	yieldContract: L563-L579

[MEDIUM] Description:

The linked calculations are prone to rounding errors, meaning it would be possible for a miniscule amount of time to pass before MXX is minted with a zero burn fee, thus allowing an attacker to spam this function and mint small albeit infinite MXX amounts repeatedly with the same set of funds.

Recommendations:

We advise the team to introduce unit tests to determine at which point this is possible and perhaps a minimum “yield” time should be imposed to alleviate the issue completely.

Alleviations:

The team opted to take our recommendation into account and used `SafeMath` operations to calculate the amounts, as a minimal precaution to this problem.

Exhibit 27

TITLE	TYPE	SEVERITY	LOCATION
Usage of `call`	Volatile Code	Informational	yieldContract: L585, L659

[INFORMATIONAL] Description:

The argument that is passed to the `call` invocation represents the signature of a function in ABI form, meaning "Collateral Return" is invalid and may cause unintended consequences when interacting with specific contracts.

Recommendations:

We advise that an empty argument is passed to `call` instead of "Collateral Return".

Alleviations:

The team opted to take our recommendation into account and used the `call` function with an empty argument.

Exhibit 28

TITLE	TYPE	SEVERITY	LOCATION
Whale Attack Vector	Volatile Code	Medium	yieldContract: L598-L602

[MEDIUM] Description:

Due to the way the contract works, it would theoretically be possible for a whale to “freeze” any MXX minting by creating a yield contract with a very high amount of funds and redeeming it instantly. This would mean that no further MXX can be minted and only another person with an equal amount of funds would be able to “unfreeze” the yield contract, causing MXX to no longer be minted if the funds used are significantly high enough.

Recommendations:

This case is particularly tricky to solve in this case. A solution at first glance would be to allow contracts to expire if they are left un-acquired and their `endTime` has passed, however this would still leave the MXX minting process frozen for the duration of the original yield which can even be equal to 270 days obliterating the protocol.

Alleviations:

No alleviations.

Exhibit 29

TITLE	TYPE	SEVERITY	LOCATION
Redundant `require` Check	Volatile Code	Informational	yieldContract: L618

[INFORMATIONAL] Description:

As the status `openMarket` is only assigned in the `earlyRedeemContract` function along with setting the `contractOwner` to `address(0)`, this `require` check will always evaluate true if the preceding `require` check evaluates true, meaning it is redundant.

Recommendations:

We advise the team to remove the redundant code.

Alleviations:

The team opted to take our recommendation into account and removed the redundant code.

Exhibit 30

TITLE	TYPE	SEVERITY	LOCATION
OpenZeppelin ERC20 Adaptation	Coding Style & Optimization	Informational	MXXERC20: L210-L211

[INFORMATIONAL] Description:

This contract appears to be identical to the OpenZeppelin implementation barring for the commented out `require` checks of the `transfer` function that allow transfers to the zero address. The purpose of this is undocumented and as such we advise it to be so.

Additionally, it would potentially be wiser to instead directly inherit the OpenZeppelin library and override the desired methods than copy-pasting the source.

Recommendations:

We advise the team to adopt the OpenZeppelin library.

Alleviations:

The team opted to take our recommendation into account and adopted the OpenZeppelin library into their codebase.

Exhibit 31

TITLE	TYPE	SEVERITY	LOCATION
Contact Implementation	Coding Style	Informational	BasicAttentionToken.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. The rationale behind these implementations is however ambiguous and we would appreciate additional comments as they are simple ERC20 tokens that directly assign the total supply to the `msg.sender`.

Recommendations:

We advise the team to revise the contract implementation.

Alleviations:

No alleviations.

Exhibit 32

TITLE	TYPE	SEVERITY	LOCATION
Contact Implementation	Coding Style	Informational	ChainLink.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. The rationale behind these implementations is however ambiguous and we would appreciate additional comments as they are simple ERC20 tokens that directly assign the total supply to the `msg.sender`. Additionally, the circulating supply differs greatly from the total supply of this token hinting to a type of locking mechanism.

Recommendations:

We advise the team to revise the contract implementation.

Alleviations:

No alleviations.

Exhibit 33

TITLE	TYPE	SEVERITY	LOCATION
Contact Implementation	Coding Style	Informational	HuobiToken.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. The rationale behind these implementations is however ambiguous and we would appreciate additional comments as they are simple ERC20 tokens that directly assign the total supply to the `msg.sender`. Additionally, the total supply defined in the contract is incorrect as the token has a total supply of 500.000.000 whilst the contract mints 5.000.000.

Recommendations:

We advise the team to revise the contract implementation.

Alleviations:

No alleviations.

Exhibit 34

TITLE	TYPE	SEVERITY	LOCATION
Contact Implementation	Coding Style	Informational	Tether.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. The rationale behind these implementations is however ambiguous and we would appreciate additional comments as they are simple ERC20 tokens that directly assign the total supply to the `msg.sender`. Additionally, the total supply defined in the contract is incorrect as the token has a dynamic total supply far greater than the defined one.

Recommendations:

We advise the team to revise the contract implementation.

Alleviations:

No alleviations.

Exhibit 35

TITLE	TYPE	SEVERITY	LOCATION
Contact Implementation	Coding Style	Informational	VeChain.sol

[INFORMATIONAL] Description:

No findings were identified at this stage of the audit. The rationale behind these implementations is however ambiguous and we would appreciate additional comments as they are simple ERC20 tokens that directly assign the total supply to the `msg.sender`. Additionally, the total supply defined in the contract as well as its abbreviation are incorrect as the token has a total supply of 86.712.634.466 whilst the contract mints 1.000.000.000 and the token has an abbreviation of `VET` whilst the contract in question uses `VEN`.

Recommendations:

We advise the team to revise the contract implementation.

Alleviations:

No alleviations.