Muse Finance

Wrap Function

Smart Contract Audit Report

June 25, 2021

IMMUNE BYTES

Introduction

1. About Muse Finance

Muse.Finance will link non ERC-20 assets with ERC-20 ecosystem, allowing the owners of staked assets on platforms such as Cosmos, IRISnet, Cardano, etc. to participate in lending, liquidity mining, yield farming which will also benefit staking service providers for generating more revenue lines.
Muse.Finance is planning to issue a liquidity token called mToken (originated by Muse.Finance) to construct a pool with DEX which will allow the users to get the yield generating assets like mATOM, mADA, mIRIS, etc. carrying unit value and able to yield farming with the Muse.Finance platform.

In order to accomplish this, we will build bridges between the Ethereum network and other networks. We will then issue wrapped Token (ERC-20 token) in the Ethereum network to registered Ethereum addresses of users who stake ATOM, ADA, IRIS, etc. in each of the networks.
Visit https://musefinance.io/ to know more about.

2. About ImmuneBytes

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

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

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

Documentation Details

The Muse Finance team has provided the following doc for the purpose of audit:

1. Concept Paper
2. The Product
3. Muse. Finance Specification V1.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

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

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Muse Finance - Wrap Contracts
- Languages: Solidity(Smart contract)
- Github commit hash for audit: 6f32fa955b9cb4b340b385fbf0b3a4fc16b2ab2e
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

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

High severity issues will bring problems and should be fixed.

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

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

Issues	High	Medium	Low
Open	No Issues Found	2	9
Closed		-	-

High severity issues

No Issues Found

Medium severity issues

1. The following functions should be declared external as they are not used anywhere else in the contract. This saves gas on function call and contract deployment.
   DefiIssuerMultiSig:
   - *implementation*
   - *setImplementation*
   - *setThreshold*
   - *addValidator*
   - *removeValidator*
   - *replaceValidator*
   - *submitMint*
   - *getValidators*
   - *getMintTxConfirmations*
   - *transactionInfoOf*

   DefiRewardMultiSig:
   - *implementation*
   - *setImplementation*
   - *setThreshold*
   - *addValidator*
   - *removeValidator*
   - *replaceValidator*
   - *getValidators*
   - *getMintTxConfirmations*
   - *transactionInfoOf*
   - *submitStakingReward*

   DefiHelper.sol:
   - *hash*
   - *hashRegister*

   DefiERC20.sol:
   - *totalBalanceOf*

   DefiWrap.sol:

- *erc20AddressOf*
- *erc20ImplementationOf*
- *issuer*
- *giver*
- *destinationAddressOf*
- *sourceAddressOf*

ERC20Burnable.sol:

- *addBurner*
- *renounceBurner*
- *burn*

2. In contract DefiWrap.sol we recommend having onlyOwner update functions for addresses _issuer and _giver. This provides the flexibility to update these addresses in the DefiWrap contract in case there is a need to update the logic of the multisig contracts interacting with it. In the current implementation since these addresses cannot be updated, a whole new DefiWrap contract has to be deployed to support new Multisig contracts and this could result in loss of existing storage. It is important to ensure that the smart contracts are future-proof and logic changes can be seamlessly integrated.

Low severity issues

1. The pragma versions used within the contracts are not locked. Consider using the latest versions among 0.5.16 or 0.5.17 for deploying the contracts as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

   ```
   pragma solidity ^0.5.0; // bad: compiles between 0.5.0 and 0.5.17
   pragma solidity 0.5.0; // good : compiles w 0.5.0 only but not the latest version
   pragma solidity 0.5.17; // best: compiles w 0.5.17
   ```

2. In contract DefiERC20.sol, line 33:

   ```
   _rate = 1 * _rate_decimal;
   ```

   This can be moved to line 16 where _rate is defined:

   ```
   uint256 public _rate;
   ```

   can be modified to:

```
uint256 private constant _rate_decimal = 1000000000000000000; //1e18
uint256 public _rate = 1 * _rate_decimal;
```

This is because _rate is not dynamically defined in the constructor and hence does not need to be there. This will save minor gas costs on deployment. Also, it is good practice to initialize all state variables for better readability and understanding.

3. In contract DefiERC20.sol, line 50:

   The function _claimReward does not need to have a non-reentrancy clause as there is no interaction with any external contract. The possibility of a reentrancy attack only arises if a contract function is trying to interact with any untrusted 3rd party contract which is not the case here as the function does only state manipulation. Removing the reentrancy clause will save gas both on deployment and interaction.

4. In contract DefiERC20.sol,
   We recommend creating another function _transferHiddenToken:

```
function _transferHiddenToken(address from, address to, uint256
amount) private returns (bool){
        uint256 _h_amount= (amount.mul(_rate_decimal)).div(_rate);
        _h_balances[to] = _h_balances[to].add(_h_amount);
        _h_balances[from] = _h_balances[from].sub(_h_amount);
        return true;
 }
```

   This will reduce gas in transfer and transferFrom functions that call both _addHiddenToken and _removeHiddenToken functions which have redundant memory usage and state changes by bundling both into the above-recommended function.

5. In contract DefiWrap.sol, line 94 and line 105

   We recommend adding a greater than zero check for the **amount** value passed as a parameter to **submitMint** and **submitBurn** functions. This is because the functions will be executed nonetheless successfully even if amount == 0 is provided with no change in the state resulting in unnecessary loss of gas.

6. There are multiple instances in the contract where the array length is calculated inside the for loop definition. As an example:

```
114     /// @dev Allows to remove an owner. Transaction has to be sent by wallet.
115     /// @param validator Address of owner.
116     function removeValidator(address validator)
117         public
118         onlyOwner
119         validatorExists(validator)
120     {
121         isValidator[validator] = false;
122         for (uint i=0; i<validators.length - 1; i++)
123             if (validators[i] == validator) {
124             validators[i] = validators[validators.length - 1];
125             break;
126         }
127         validators.length -= 1;
128         changeRequirement();
129     }
```

In such cases, array.length is called each time the for loop is executed which uses a lot of gas.

We recommend storing the array length in a variable before the for loop and use that variable in the for loop definition. This ensures array length is calculated only once which saves gas.

The following functions can be improved:

- *Contract DefiIssuerMultiSig.sol, line 122 (removeValidator), 140 (replaceValidator)*
- *Contract DefiRewardMultiSig.sol, line 104 (removeValidator), 122 (replaceValidator)*

7. Contract DefiRewardMultisig.sol, line 261:
   We recommend modifying getMintTxConfirmations to getSubmitStakingRewardTxConfirmations (or similar) to align with the functionality and naming conventions of other functions in the contract.

8. Contract DefiIssuerMultiSig, line 155
   Contract DefiRewardMultiSig, line 135

   We recommend using safemath for the arithmetic operation. Although it is a fairly simple arithmetic operation, safemath adds an extra layer of security over overflow/underflow issues.

```
function changeRequirement()
    internal
{
    if(!fixedThreshold){
        uint _required = (validators.length * uint(2)) / uint(3);
        if (_required == 0)
            _required = 1;
        validatorThreshold = _required;
    }
}
```

9. Contract DefiERC20.sol, line 45:

The function claimReward() expects a return value for unit256 userReward but none is returned and hence raises a compilation warning:

```
DefiERC20.sol:45:46: Warning:
Unused function parameter. Remove
or comment out the variable name
to silence this warning. function
claimReward() external returns
(uint256 userReward){ ^----------
-----^
```

We recommend to return the result of _claimReward to silence this warning. Either of these will work in the claimReward function:

```
userReward = _claimReward(msg.sender);
```

Or:

```
return _claimReward(msg.sender);
```

Recommendations

1. Add events for most state changes. For example, when creating new DefiERC20 tokens from DefiWrap, when registering a new sourceAddress or claiming rewards in DefiERC20. Events should be fired with all state variable updates as good practice. This makes the contract future proof for usage in frontend applications and event listener backend services.

2. Follow solidity style guide for better readability:
   https://docs.soliditylang.org/en/v0.7.5/style-guide.html.
   For example, Functions should be grouped according to their visibility and ordered:
   - constructor
   - receive function (if exists)
   - fallback function (if exists)
   - external
   - public
   - internal
   - private
   - Within a grouping, place the view and pure functions last.

   Note: Linting violations can be easily fixed using linters like solhint.

3. Add Natspecs comments to all functions for a better understanding regarding what the parameters mean and what the function does.

4. All the "*require*" statements used in the contract should also specify error messages for easy debugging.

5. We recommend following the solidity naming conventions. There are a lot of naming in the contracts that deviate from the conventions. For example, *_enableStake_Bonus* does not follow a mixed case format.

6. We recommend avoiding using literals with too many digits as it is hard to read and could easily be mistaken. For example,

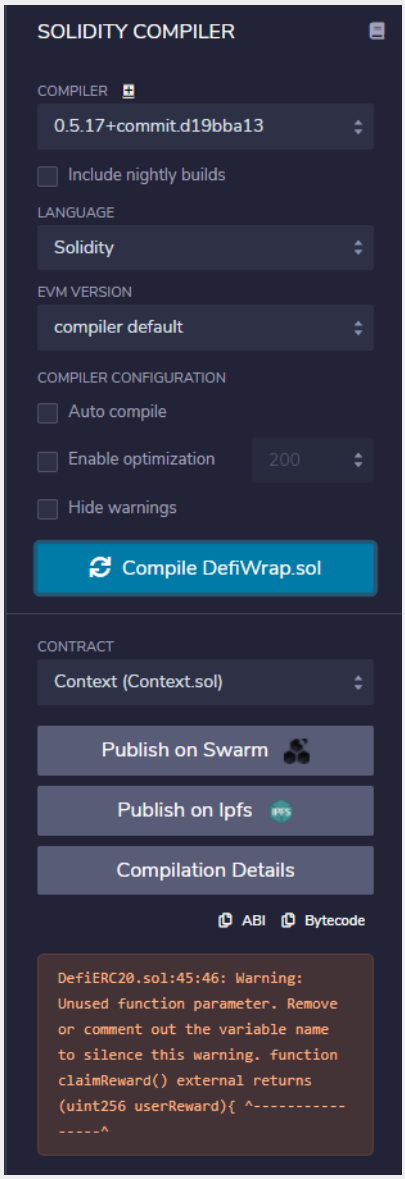
   ```
   uint256 private constant _rate_decimal = 1000000000000000000; //1e18
   ```

   We recommend using the Ether suffix.

Automated Audit

Remix Compiler Warnings

One warning was thrown by the compiler which is covered in the manual audit:



Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real-time.

We performed analysis using the contract Library on the mainnet address of the DefiWrap contract on kovan: 0xa8446884a2790BaD19cA0424446987659E68d700

Analysis summary can be accessed here: https://contract-library.com/contracts/Kovan/0xA8446884A2790BAD19CA0424446987659E68D700

It did not return any issue during the analysis.

SmartCheck

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

```
ruleId: SOLIDITY_PRAGMAS_VERSION
patternId: 23fc32
severity: 1
line: 1
column: 16
content: ^

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 17
column: 34
content: private

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
```

```
severity: 1
line: 18
column: 35
content: private

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 22
column: 32
content: private

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 23
column: 52
content: private

ruleId: SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA
patternId: 5616b2
severity: 1
line: 26
column: 51
content: private

ruleId: SOLIDITY_VISIBILITY
patternId: 910067
severity: 1
line: 58
column: 4
content: functioncreateDefiERC20(stringcalldataname,stringcalldata<missing
')'>

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 58
column: 67
content: symbol,
```

```
ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 58
column: 75
content: uint8decimals,

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 58
column: 91
content: uint256cap)external

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 60
column: 8
content:
onlyOwner{DefiERC20_defiERC20=newDefiERC20(name,symbol,decimals,cap);

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 63
column: 7
content: _erc20Names[symbol]=_defiERC20;

ruleId: SOLIDITY_VISIBILITY
patternId: b51ce0
severity: 1
line: 64
column: 7
content: _erc20Addresses[address(_defiERC20)]=_defiERC20;

SOLIDITY_VISIBILITY :7
SOLIDITY_PRAGMAS_VERSION :1
SOLIDITY_PRIVATE_MODIFIER_DONT_HIDE_DATA :5
```

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

Slither

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

```
➜  contracts slither DefiIssuerMultiSig.sol
INFO:Detectors:
Reentrancy in
DefiIssuerMultiSig.executeMintTransaction(address,bytes32,bytes32)
(DefiIssuerMultiSig.sol#187-207):
        External calls:
        - ! external_mint(erc20Address,hashSourceAddress,_amount)
(DefiIssuerMultiSig.sol#202)
                - result =
_implementation.submitMint(erc20Address,hashSourceAddress,amount)
(DefiIssuerMultiSig.sol#211)
        State variables written after the call(s):
        - transactions[transactionId][hashSourceAddress] = _txData
(DefiIssuerMultiSig.sol#205)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vu
lnerabilities-1
INFO:Detectors:
Context._msgData() (@openzeppelin/contracts/GSN/Context.sol#23-26) is never
used and should be removed
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.5.0 (@openzeppelin/contracts/GSN/Context.sol#1) allows old
versions
Pragma version^0.5.0 (@openzeppelin/contracts/ownership/Ownable.sol#1)
allows old versions
Pragma version^0.5.0 (DefiHelper.sol#1) allows old versions
Pragma version^0.5.0 (DefiIssuerMultiSig.sol#1) allows old versions
```

```
Pragma version^0.5.0 (IDefiWrap.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter DefiIssuerMultiSig.setThreshold(bool,uint256)._fixed
(DefiIssuerMultiSig.sol#91) is not in mixedCase
Function DefiIssuerMultiSig.external_mint(address,bytes32,uint256)
(DefiIssuerMultiSig.sol#209-213) is not in mixedCase
Parameter
DefiIssuerMultiSig.transactionInfoOf(address,bytes32,bytes32,uint256)._erc20Address (DefiIssuerMultiSig.sol#279) is not in mixedCase
Parameter
DefiIssuerMultiSig.transactionInfoOf(address,bytes32,bytes32,uint256)._amount (DefiIssuerMultiSig.sol#279) is not in mixedCase
Variable DefiIssuerMultiSig._implementation (DefiIssuerMultiSig.sol#26) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (@openzeppelin/contracts/GSN/Context.sol#24)"
inContext (@openzeppelin/contracts/GSN/Context.sol#13-27)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner()
(@openzeppelin/contracts/ownership/Ownable.sol#30-32)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership()
(@openzeppelin/contracts/ownership/Ownable.sol#56-59)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address)
(@openzeppelin/contracts/ownership/Ownable.sol#65-67)
hash(string) should be declared external:
        - DefiHelper.hash(string) (DefiHelper.sol#4-9)
hashRegister(address,bytes32) should be declared external:
```

```
        - DefiHelper.hashRegister(address,bytes32) (DefiHelper.sol#11-16)
implementation() should be declared external:
        - DefiIssuerMultiSig.implementation()
(DefiIssuerMultiSig.sol#79-81)
setImplementation(address) should be declared external:
        - DefiIssuerMultiSig.setImplementation(address)
(DefiIssuerMultiSig.sol#87-89)
setThreshold(bool,uint256) should be declared external:
        - DefiIssuerMultiSig.setThreshold(bool,uint256)
(DefiIssuerMultiSig.sol#91-100)
addValidator(address) should be declared external:
        - DefiIssuerMultiSig.addValidator(address)
(DefiIssuerMultiSig.sol#104-113)
removeValidator(address) should be declared external:
        - DefiIssuerMultiSig.removeValidator(address)
(DefiIssuerMultiSig.sol#117-130)
replaceValidator(address,address) should be declared external:
        - DefiIssuerMultiSig.replaceValidator(address,address)
(DefiIssuerMultiSig.sol#135-150)
submitMint(address,bytes32,bytes32,uint256) should be declared external:
        - DefiIssuerMultiSig.submitMint(address,bytes32,bytes32,uint256)
(DefiIssuerMultiSig.sol#165-173)
getValidators() should be declared external:
        - DefiIssuerMultiSig.getValidators()
(DefiIssuerMultiSig.sol#251-256)
getMintTxConfirmations(bytes32) should be declared external:
        - DefiIssuerMultiSig.getMintTxConfirmations(bytes32)
(DefiIssuerMultiSig.sol#261-277)
transactionInfoOf(address,bytes32,bytes32,uint256) should be declared
external:
        -
DefiIssuerMultiSig.transactionInfoOf(address,bytes32,bytes32,uint256)
(DefiIssuerMultiSig.sol#279-288)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-functi
on-that-could-be-declared-external
INFO:Slither:DefiIssuerMultiSig.sol analyzed (5 contracts with 75
detectors), 29 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors
and Github integration
```

```
➜  contracts slither DefiRewardMultiSig.sol
INFO:Detectors:
Reentrancy in DefiRewardMultiSig.executeSubmitTransaction(address,bytes32)
(DefiRewardMultiSig.sol#168-191):
        External calls:
        - ! external_submit(erc20Address,_amount)
(DefiRewardMultiSig.sol#183)
                - result =
_implementation.submitStakingReward(erc20Address,amount)
(DefiRewardMultiSig.sol#195)
        State variables written after the call(s):
        - transactions[transactionId] = _txData
(DefiRewardMultiSig.sol#189)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vu
lnerabilities-1
INFO:Detectors:
Context._msgData() (@openzeppelin/contracts/GSN/Context.sol#23-26) is never
used and should be removed
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.5.0 (@openzeppelin/contracts/GSN/Context.sol#1) allows old
versions
Pragma version^0.5.0 (@openzeppelin/contracts/ownership/Ownable.sol#1)
allows old versions
Pragma version^0.5.0 (DefiHelper.sol#1) allows old versions
Pragma version^0.5.0 (DefiRewardMultiSig.sol#1) allows old versions
Pragma version^0.5.0 (IDefiWrap.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-ver
sions-of-solidity
INFO:Detectors:
Parameter DefiRewardMultiSig.setThreshold(bool,uint256)._fixed
(DefiRewardMultiSig.sol#72) is not in mixedCase
Function DefiRewardMultiSig.external_submit(address,uint256)
(DefiRewardMultiSig.sol#193-197) is not in mixedCase
Parameter
DefiRewardMultiSig.transactionInfoOf(address,uint256,uint256)._erc20Address
```

```
(DefiRewardMultiSig.sol#281) is not in mixedCase
Parameter
DefiRewardMultiSig.transactionInfoOf(address,uint256,uint256)._amount
(DefiRewardMultiSig.sol#281) is not in mixedCase
Variable DefiRewardMultiSig._implementation (DefiRewardMultiSig.sol#24) is
not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-t
o-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (@openzeppelin/contracts/GSN/Context.sol#24)"
inContext (@openzeppelin/contracts/GSN/Context.sol#13-27)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-sta
tements
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner()
(@openzeppelin/contracts/ownership/Ownable.sol#30-32)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership()
(@openzeppelin/contracts/ownership/Ownable.sol#56-59)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address)
(@openzeppelin/contracts/ownership/Ownable.sol#65-67)
hash(string) should be declared external:
        - DefiHelper.hash(string) (DefiHelper.sol#4-9)
hashRegister(address,bytes32) should be declared external:
        - DefiHelper.hashRegister(address,bytes32) (DefiHelper.sol#11-16)
setThreshold(bool,uint256) should be declared external:
        - DefiRewardMultiSig.setThreshold(bool,uint256)
(DefiRewardMultiSig.sol#72-81)
addValidator(address) should be declared external:
        - DefiRewardMultiSig.addValidator(address)
(DefiRewardMultiSig.sol#85-94)
removeValidator(address) should be declared external:
        - DefiRewardMultiSig.removeValidator(address)
(DefiRewardMultiSig.sol#98-112)
replaceValidator(address,address) should be declared external:
        - DefiRewardMultiSig.replaceValidator(address,address)
```

```
(DefiRewardMultiSig.sol#117-131)
submitStakingReward(address,uint256,uint256) should be declared external:
        - DefiRewardMultiSig.submitStakingReward(address,uint256,uint256)
(DefiRewardMultiSig.sol#145-154)
implementation() should be declared external:
        - DefiRewardMultiSig.implementation()
(DefiRewardMultiSig.sol#239-241)
setImplementation(address) should be declared external:
        - DefiRewardMultiSig.setImplementation(address)
(DefiRewardMultiSig.sol#247-249)
getValidators() should be declared external:
        - DefiRewardMultiSig.getValidators()
(DefiRewardMultiSig.sol#253-258)
getMintTxConfirmations(bytes32) should be declared external:
        - DefiRewardMultiSig.getMintTxConfirmations(bytes32)
(DefiRewardMultiSig.sol#263-279)
transactionInfoOf(address,uint256,uint256) should be declared external:
        - DefiRewardMultiSig.transactionInfoOf(address,uint256,uint256)
(DefiRewardMultiSig.sol#281-290)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-functi
on-that-could-be-declared-external
INFO:Slither:DefiRewardMultiSig.sol analyzed (5 contracts with 75
detectors), 29 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors
and Github integration
➜  contracts slither DefiWrap.sol
INFO:Detectors:
DefiWrap.submitStakingReward(address,uint256) (DefiWrap.sol#89-94) ignores
return value by IDefiERC20(erc20Address).submitStakingReward(amount)
(DefiWrap.sol#92)
DefiWrap.submitMint(address,bytes32,uint256) (DefiWrap.sol#97-106) ignores
return value by
IDefiERC20(erc20Address).submitMint(_registeredAddresses[transactionId],amo
unt) (DefiWrap.sol#104)
DefiWrap.submitBurn(address,uint256) (DefiWrap.sol#108-115) ignores return
value by _erc20Addresses[erc20Address].submitBurn(msg.sender,amount)
(DefiWrap.sol#112)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
INFO:Detectors:
ERC20Capped.constructor(uint256).cap
(@openzeppelin/contracts/token/ERC20/ERC20Capped.sol#15) shadows:
        - ERC20Capped.cap()
(@openzeppelin/contracts/token/ERC20/ERC20Capped.sol#23-25) (function)
ERC20Detailed.constructor(string,string,uint8).name
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#18) shadows:
        - ERC20Detailed.name()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#27-29) (function)
ERC20Detailed.constructor(string,string,uint8).symbol
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#18) shadows:
        - ERC20Detailed.symbol()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#35-37) (function)
ERC20Detailed.constructor(string,string,uint8).decimals
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#18) shadows:
        - ERC20Detailed.decimals()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#51-53) (function)
DefiERC20.constructor(string,string,uint8,uint256).name (DefiERC20.sol#35)
shadows:
        - ERC20Detailed.name()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#27-29) (function)
DefiERC20.constructor(string,string,uint8,uint256).symbol
(DefiERC20.sol#36) shadows:
        - ERC20Detailed.symbol()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#35-37) (function)
DefiERC20.constructor(string,string,uint8,uint256).decimals
(DefiERC20.sol#37) shadows:
        - ERC20Detailed.decimals()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#51-53) (function)
DefiERC20.constructor(string,string,uint8,uint256).cap (DefiERC20.sol#38)
shadows:
        - ERC20Capped.cap()
(@openzeppelin/contracts/token/ERC20/ERC20Capped.sol#23-25) (function)
DefiWrap.constructor(address,address).issuer (DefiWrap.sol#55) shadows:
        - DefiWrap.issuer() (DefiWrap.sol#125-127) (function)
DefiWrap.constructor(address,address).giver (DefiWrap.sol#55) shadows:
        - DefiWrap.giver() (DefiWrap.sol#129-131) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

```
INFO:Detectors:
DefiERC20.submitBurn(address,uint256) (DefiERC20.sol#91-100) should emit an
event for:
        - _deposited_total = _deposited_total.sub(amount)
(DefiERC20.sol#97)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-event
s-arithmetic
INFO:Detectors:
DefiWrap.constructor(address,address).issuer (DefiWrap.sol#55) lacks a
zero-check on :
                - _issuer = issuer (DefiWrap.sol#56)
DefiWrap.constructor(address,address).giver (DefiWrap.sol#55) lacks a
zero-check on :
                - _giver = giver (DefiWrap.sol#57)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-
address-validation
INFO:Detectors:
Reentrancy in DefiWrap.submitBurn(address,uint256) (DefiWrap.sol#108-115):
        External calls:
        - _erc20Addresses[erc20Address].submitBurn(msg.sender,amount)
(DefiWrap.sol#112)
        Event emitted after the call(s):
        - BurnConfirmation(msg.sender,erc20Address,amount)
(DefiWrap.sol#113)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vu
lnerabilities-3
INFO:Detectors:
Context._msgData() (@openzeppelin/contracts/GSN/Context.sol#23-26) is never
used and should be removed
ERC20._burnFrom(address,uint256)
(@openzeppelin/contracts/token/ERC20/ERC20.sol#226-229) is never used and
should be removed
SafeMath.mod(uint256,uint256)
(@openzeppelin/contracts/math/SafeMath.sol#135-137) is never used and
should be removed
SafeMath.mod(uint256,uint256,string)
(@openzeppelin/contracts/math/SafeMath.sol#152-155) is never used and
```

```
should be removed
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.5.0 (@openzeppelin/contracts/GSN/Context.sol#1) allows old
versions
Pragma version^0.5.0 (@openzeppelin/contracts/access/Roles.sol#1) allows
old versions
Pragma version^0.5.0
(@openzeppelin/contracts/access/roles/MinterRole.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts/math/SafeMath.sol#1) allows
old versions
Pragma version^0.5.0 (@openzeppelin/contracts/ownership/Ownable.sol#1)
allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts/token/ERC20/ERC20.sol#1)
allows old versions
Pragma version^0.5.0
(@openzeppelin/contracts/token/ERC20/ERC20Capped.sol#1) allows old versions
Pragma version^0.5.0
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#1) allows old
versions
Pragma version^0.5.0
(@openzeppelin/contracts/token/ERC20/ERC20Mintable.sol#1) allows old
versions
Pragma version^0.5.0 (@openzeppelin/contracts/token/ERC20/IERC20.sol#1)
allows old versions
Pragma version^0.5.0 (DefiERC20.sol#1) allows old versions
Pragma version^0.5.0 (DefiHelper.sol#1) allows old versions
Pragma version^0.5.0 (DefiWrap.sol#1) allows old versions
Pragma version^0.5.0 (ERC20Burnable.sol#1) allows old versions
Pragma version^0.5.0 (IDefiERC20.sol#1) allows old versions
Pragma version^0.5.0 (IDefiWrap.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-ver
sions-of-solidity
INFO:Detectors:
Variable DefiERC20._h_balances (DefiERC20.sol#16) is not in mixedCase
Variable DefiERC20._h_totalSupply (DefiERC20.sol#18) is not in mixedCase
Variable DefiERC20._deposited_total (DefiERC20.sol#19) is not in mixedCase
```

```
Variable DefiERC20._rate (DefiERC20.sol#22) is not in mixedCase
Constant DefiERC20._rate_decimal (DefiERC20.sol#23) is not in
UPPER_CASE_WITH_UNDERSCORES
Variable DefiWrap._issuer (DefiWrap.sol#30) is not in mixedCase
Variable DefiWrap._giver (DefiWrap.sol#32) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (@openzeppelin/contracts/GSN/Context.sol#24)"
inContext (@openzeppelin/contracts/GSN/Context.sol#13-27)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
DefiERC20.slitherConstructorConstantVariables() (DefiERC20.sol#9-169) uses
literals with too many digits:
        - _rate_decimal = 1000000000000000000 (DefiERC20.sol#23)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
addMinter(address) should be declared external:
        - MinterRole.addMinter(address)
(@openzeppelin/contracts/access/roles/MinterRole.sol#27-29)
renounceMinter() should be declared external:
        - MinterRole.renounceMinter()
(@openzeppelin/contracts/access/roles/MinterRole.sol#31-33)
owner() should be declared external:
        - Ownable.owner()
(@openzeppelin/contracts/ownership/Ownable.sol#30-32)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership()
(@openzeppelin/contracts/ownership/Ownable.sol#56-59)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address)
(@openzeppelin/contracts/ownership/Ownable.sol#65-67)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address)
(@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
```

```
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256)
(@openzeppelin/contracts/token/ERC20/ERC20.sol#81-84)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256)
(@openzeppelin/contracts/token/ERC20/ERC20.sol#116-119)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256)
(@openzeppelin/contracts/token/ERC20/ERC20.sol#135-138)
cap() should be declared external:
        - ERC20Capped.cap()
(@openzeppelin/contracts/token/ERC20/ERC20Capped.sol#23-25)
name() should be declared external:
        - ERC20Detailed.name()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#27-29)
symbol() should be declared external:
        - ERC20Detailed.symbol()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#35-37)
decimals() should be declared external:
        - ERC20Detailed.decimals()
(@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol#51-53)
totalBalanceOf(address) should be declared external:
        - DefiERC20.totalBalanceOf(address) (DefiERC20.sol#116-118)
erc20AddressOf(string) should be declared external:
        - DefiWrap.erc20AddressOf(string) (DefiWrap.sol#117-119)
erc20ImplementationOf(address) should be declared external:
        - DefiWrap.erc20ImplementationOf(address) (DefiWrap.sol#121-123)
issuer() should be declared external:
        - DefiWrap.issuer() (DefiWrap.sol#125-127)
giver() should be declared external:
        - DefiWrap.giver() (DefiWrap.sol#129-131)
destinationAddressOf(address,bytes32) should be declared external:
        - DefiWrap.destinationAddressOf(address,bytes32)
(DefiWrap.sol#133-136)
sourceAddressOf(address,address) should be declared external:
        - DefiWrap.sourceAddressOf(address,address) (DefiWrap.sol#138-141)
addBurner(address) should be declared external:
        - BurnerRole.addBurner(address) (ERC20Burnable.sol#27-29)
renounceBurner() should be declared external:
        - BurnerRole.renounceBurner() (ERC20Burnable.sol#31-33)
```

```
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (ERC20Burnable.sol#47-49)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:DefiWrap.sol analyzed (17 contracts with 75 detectors), 70 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither did not raise any high severity issues. All other issues are covered in the manual audit or are not relevant.

Concluding Remarks

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

Our auditors suggest that Medium and Low severity issues should be resolved by the developers. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Muse Finance platform or its product nor this audit is investment advice.
Notes:

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

ImmuneBytes Pvt Ltd.