// HALBORN

Yieldly.Finance

Bridge Ethereum Smart Contract Audit

Prepared by: **Halborn**

Date of Engagement: **May 19th-31st, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 5
CONTACTS 5
1 EXECUTIVE OVERVIEW 6
1.1 INTRODUCTION 7
1.2 AUDIT SUMMARY 7
1.3 TEST APPROACH & METHODOLOGY 8
RISK METHODOLOGY 9
1.4 SCOPE 11
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 12
3 FINDINGS & TECH DETAILS 13
3.1 (HAL-01) LACK OF AMOUNT CHECK - MEDIUM 15
Description 15
Code Location 15
Risk Level 16
Recommendation 16
Remediation Plan 16
3.2 (HAL-02) INTEGER OVERFLOW - MEDIUM 17
Description 17
Code Location 17
Risk Level 19
Recommendation 19
Remediation Plan 20
3.3 (HAL-03) MISSING ADDRESS VALIDATION - LOW 22
Description 22

Code Location 22
Risk Level 22
Recommendation 23
Remediation Plan 23
3.4 (HAL-04) THRESHOLD LIMIT VALIDATION MISSING - LOW 24
Description 24
Function 24
Risk Level 24
Recommendation 24
Remediation Plan 25
3.5 (HAL-05) FLOATING PRAGMA - LOW 26
Description 26
Code Location 26
Risk Level 26
Recommendation 26
Remediation Plan 27
3.6 (HAL-06) OWNER CAN RENOUNCE OWNERSHIP - LOW 29
Description 29
Function 29
Risk Level 29
Recommendation 30
Remediation Plan 30
3.7 (HAL-07) MISSING EVENT HANDLER - LOW 31
Description 31
Code Location 31
Risk Level 31

Recommendation 31
Remediation Plan 32
3.8 (HAL-08) IGNORE RETURN VALUES - LOW 33
Description 33
Code Location 33
Risk Level 34
Recommendation 34
Remediation Plan 34
3.9 (HAL-09) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 36
Description 36
Code Location 36
Risk Level 38
Recommendation 38
Remediation Plan 38
3.10 (HAL-10) BLOCK TIMESTAMP USAGE - INFORMATIONAL 39
Description 39
Code Location 39
Risk Level 39
Recommendation 39
Remediation plan 40
3.11 (HAL-11) REDUNDANT BOOLEAN COMPARISON - INFORMATIONAL 41
Description 41
Code Location 41
Risk Level 41
Recommendation 42

Remediation plan 42

3.12 (HAL-12) FOR LOOP OVER DYNAMIC ARRAY - INFORMATIONAL 43

Description 43

Example Location 43

Risk Level 44

Recommendation 44

Remediation plan 44

3.13 STATIC ANALYSIS REPORT 46

Description 46

Results 46

3.14 AUTOMATED SECURITY SCAN RESULTS 53

Description 53

Results 53

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/29/2021	Gokberk Gulgun
0.2	Document Edits	05/30/2021	Gabi Urrutia
1.0	Final Version	05/31/2021	Gokberk Gulgun
1.1	Remediation Plan	06/07/2021	Gokberk Gulgun

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Yieldly.Finance bridge component is designed to integrate a diverse set of blockchains specialised for different needs. Yieldly.Finance connects Algorand to Ethereum , and vice versa.

Yieldly.Finance engaged Halborn to conduct a security assessment on their Smart contract beginning on May 19th, 2021 and ending May 31th, 2021. The security assessment was scoped to the smart contract repository. An audit of the security risk and implications regarding the changes introduced by the development team at Yieldly.Finance prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned three full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified few security risks which were solved by Yieldly.Finance team. The fixes have been reviewed by the auditors.

Risk Assessment Sheet

Bug	Status	Description
Reentrancy	PASS	One of the major dangers of calling external contracts is that they can take over the control flow. There is not issue related with a re-entrancy.
Missing Protection against Signature Replay Attacks	PASS	It was seen that the ecrerecover function was not used.
Requirement Violation	PASS	Access control policies are properly implemented.
Signature Malleability	PASS	A signature didnt include into a signed message hash. Therefore, the processed previous messages are checked in the contract.
DoS With Block Gas Limit	PASS	Actions that require looping across the entire data structure are avoided.
Hash Collisions With Multiple Variable Length Arguments	PASS	It was seen that the encodePacked function was not used.
Typographical Error	PASS	Arithmetic calculations protected by a pragma version.
Use of Deprecated Solidity Functions	PASS	Deprecated solidity functions are not used.
Authorization through tx.origin	PASS	Tx.origin authentication is not used.
Integer Overflow/Underflow	PASS	The pragma keyword is used to enable certain compiler features or checks.
Timestamp Dependence	PASS	During the test, block.timestamp usage has been observed. Block.number usage is suggested instead of block.timestamp.
Unchecked Call Return Value	PASS	In the contracts, Return values are handled by the functions.
Floating Pragma	PASS	The pragma keyword is used to enable certain compiler features or checks. The pragma locked into '0.8.4'.
Function Default Visibility	PASS	The visibility of the functions are properly implemented.
Unprotected Ether Withdrawal	PASS	The withdraw progress is completed according to access control.
Unprotected Self.destruction Instruction	PASS	It was seen that the self.destruction function was not used.
Uninitialized Storage Parameters	PASS	It was seen that the memory argument was used over the initial function in storage usage.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit.While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Static Analysis of security for scoped contract, and imported functions.(Slither)

- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Testnet deployment (Truffle, Ganache)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the smart contracts:

Commit Id: e28c66e427668a981f26068f36c00ba937339663 - https://github.com/yieldly-finance/yieldly-bridge-smart-contracts/blob/master/yieldly-bridge-eth/contracts

Dispatcher:
- Dispatcher.sol
- DispatcherMultiSig.sol

Vault:
- Vault.sol
- VaultMultiSig.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	6	4

LIKELIHOOD

IMPACT

	(HAL-01)			
	(HAL-05) (HAL-06)	(HAL-02)		
(HAL-09)	(HAL-03) (HAL-04) (HAL-07) (HAL-08)			
(HAL-10) (HAL-11) (HAL-12)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - LACK OF AMOUNT CHECK	Medium	RISK ACCEPTED: 06/02/2021
HAL02 - INTEGER OVERFLOW	Medium	SOLVED: 06/02/2021
HAL03 - MISSING ADDRESS VALIDATION	Low	SOLVED: 06/02/2021
HAL04 - THRESHOLD LIMIT VALIDATION MISSING	Low	SOLVED: 06/02/2021
HAL05 - FLOATING PRAGMA	Low	SOLVED: 06/02/2021
HAL06 - OWNER CAN RENOUNCE OWNERSHIP	Low	SOLVED: 06/02/2021
HAL07 - MISSING EVENT HANDLER	Low	SOLVED: 06/02/2021
HAL08 - IGNORE RETURN VALUES	Low	SOLVED: 06/02/2021
HAL09 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 06/02/2021
HAL10 - BLOCK TIMESTAMP USAGE	Informational	NOT APPLICABLE: 06/02/2021
HAL11 - REDUNDANT BOOLEAN COMPARISON	Informational	SOLVED: 06/02/2021
HAL12 - FOR LOOP OVER DYNAMIC ARRAY	Informational	SOLVED: 06/02/2021
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN RESULTS	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF AMOUNT CHECK - MEDIUM

Description:

In the Yieldly.Finance workflow, the function transferFunds utilizes user-supplied amount parameter to calculate the purchased amount of transferProposals. The contract however does not verify the amount actually paid to be equal to what the user declared. Vault contract transferFunds method is missing amount check. Although, Only The dispatcher can access the function, Lack of amount check still poses a risk.

In the transferFunds function Yieldly.Finance uses OpenZeppelin's transfer to handle the token transfer. However, since the actual amount transferred i.e. the delta of previous (before transfer) and current (after transfer) balance is not verified, a malicious dispatcher can pay with a custom ERC20 token with the transfer function modified in such a way that it does not update balances at all.

Code Location:

Vault.sol Line #~37

Listing 1: Vault.sol (Lines 37)

```
37     function transferFunds(address _tokenAddress, address
           _recipient, uint256 _amount) public onlyDispatcher {
38         require(tokensStore[_tokenAddress].active == true, "Token
               not supported");
39         require(_amount > 0, "Cannot transfer 0 tokens");
40         ERC20(_tokenAddress).transfer(_recipient, _amount);
41         emit ReleasedFundsEvent(_recipient, _amount);
42     }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It would be safer to perform additional validation before assigning user-supplied values on contracts. Transfer amount should be checked on the related function even though there is an access control check.

Remediation Plan:

RISK ACCEPTED: The bridge transfers tokens out of the vault upon validation from the dispatcher. Any tokens that the bridge supports are added by Yieldly.Finance Team and audited. There is only one dispatcher in the contract that is calling this function. There is also no payment being made to the vault on the ethereum side. From that reasons, Yieldly.Finance Team has decided to continue without amount validation.

3.2 (HAL-02) INTEGER OVERFLOW - MEDIUM

Description:

An overflow happens when an arithmetic operation reaches the maximum size of a type. For instance, in Dispatcher.sol the proposeNewTxn method is incrementing the number of proposal per transfer and may end up overflowing the integer since the resulting value is not checked to be lower than max allowed (2^32 -1). In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum representable value.

Code Location:

Dispatcher.sol Line #149

Listing 2: Dispatcher.sol (Lines 149)

```
131     function proposeNewTxn(address _userAddress, address
            _tokenAddress, uint256 _amount, string memory _note) public
             onlyBridgeController{
132         transferProposalStore[uuid].recipientAddress =
                _userAddress;
133         transferProposalStore[uuid].amount = _amount;
134         transferProposalStore[uuid].tokenAddress = _tokenAddress;
135         transferProposalStore[uuid].note = _note;
136         if(valThreshold == 1)
137         {
138             vault.transferFunds(transferProposalStore[uuid].
                    tokenAddress, transferProposalStore[uuid].
                    recipientAddress, transferProposalStore[uuid].
                    amount);
139             emit ApprovedTransaction(transferProposalStore[uuid].
                    recipientAddress, transferProposalStore[uuid].
                    amount, uuid);
140             emit proposalCreated(uuid);
```

```
141             transferProposalStore[uuid].signed = true;
142         }
143         else
144         {
145             transferProposalStore[uuid].signatures.push(msg.sender
                    );
146             outstandingTransferProposalsIndex.push(uuid);
147             emit proposalCreated(uuid);
148         }
149         uuid += 1;
150     }
```

Dispatcher.sol Line #120

Listing 3: Dispatcher.sol (Lines 120)

```
110  function removeValidator(address _validatorAddress) public
        onlyMultiSig {
111         //Remove a validator threshold count in order to avoid not
                having enough validators
112         for(uint256 i = 0; i <= validators.length; i++)
113         {
114             if(validators[i] == _validatorAddress)
115             {
116                 validators[i] = validators[validators.length - 1];
117                 validators.pop();
118                 if(valThreshold > 1)
119                 {
120                     valThreshold = valThreshold - 1;
121                 }
122     ...
```

VaultMultiSig.sol Line #93-108

Listing 4: VaultMultiSig.sol (Lines 93)

```
 88    function proposeAddress(address _address, uint256 _index)
          public onlyOwner {
 89         addressProposalStore[uuid].proposal = _address;
 90         addressProposalStore[uuid].proposalType = _index;
 91         addressProposalStore[uuid].timeStamp = block.timestamp;
 92         outstandingAddressProposalsIndex.push(uuid);
```

```
93          uuid += 1;
94      }
95      ...
```

VaultMultiSig.sol Line #137

Listing 5: VaultMultiSig.sol (Lines 138)

```
132     ...
133     if (addressProposalStore[_proposal].signatures.length >=
            threshold) {
134             addressProposalStore[_proposal].signed = true;
135             //Remove a threshold count in order to avoid not
                    having enough signatories
136             if(threshold > 1)
137             {
138                 threshold = threshold - 1;
139             }
140             removeSignatory(addressProposalStore[_proposal].
                    proposal);
141             popAddressProposal(_proposal);
142             emit RemovedSignatory(addressProposalStore[_proposal].
                    proposal);
143         }
144     ...
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system.

Remediation Plan:

SOLVED: In the proposeAddress and proposeNewTxn function, the UUID parameter is appended once per the proposition of a new transaction. UUID parameter defined as a uint256 - 2*256-1. The max amount of transactions that are required in order to overflow the unique identifier. Therefore, the overflow and underflow are not practically possible.

In the other hand, the threshold variable is set by a signatories. The low bound of threshold variable is checked on the function. Due to this implementation, the underflow is not possible. From the code pieces below, we can clearly see the function is only accessible from signatories. Also, the threshold parameter defined as uint256. The overflow also is not possible practically.

Listing 6: DispatcherMultiSig.sol (Lines 211,212)

```
206     function approveNewThreshold(uint256 _proposal) external
            onlySignatories oneVoteThreshold(_proposal){
207         require(thresholdProposalStore[_proposal].signed == false,
                "Already Signed");
208         require(thresholdProposalStore[_proposal].proposal <=
                signatories.length, "Can't be less signatories than
                threshold");
209         thresholdProposalStore[_proposal].signatures.push(msg.
                sender);
210
211         if (thresholdProposalStore[_proposal].signatures.length >=
                threshold) {
212             threshold = thresholdProposalStore[_proposal].proposal
                    ;
213             popThresholdProposal(_proposal);
214             emit ApprovedNewThreshold(thresholdProposalStore[
                    _proposal].proposal);
215         }
```

0.8.0 and later versions include native overflow and underflow checks that used to require the SafeMath lib or custom checks to avoid. Yieldly .Finance Team will use Pragma 0.8.4 therefore, safety mechanisms will be activated.

Listing 7: Dispatcher.sol (Lines 1)

```
1 pragma solidity 0.8.4;
2 import "./Vault.sol";
3 import "./openzeppelin_contracts/access/Ownable.sol";
4 import "./openzeppelin_contracts/token/ERC20/ERC20.sol";
```

3.3 (HAL-03) MISSING ADDRESS VALIDATION - LOW

Description:

In the Yieldly.Finance - Vault contract is missing a safety check inside their constructors and multiple functions. Setters of address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burnt forever.

Code Location:

Vault.sol Line #~18

Listing 8: Vault.sol (Lines 19)

```
18     constructor(address _multiSigAddress) Ownable() {
19         multiSigAddress = _multiSigAddress;
20     }
```

Dispatcher.sol Line #~34-36

Listing 9: Dispatcher.sol (Lines 34,35,36)

```
33  constructor(address _vaultAddress, address _multiSigAddress)
        Ownable() {
34          multiSigAddress = _multiSigAddress;
35          vault = Vault(_vaultAddress);
36          bridgeControllerAddress = msg.sender;
37      }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add proper address validation when assigning a value to a variable from user-supplied data. Better yet, address white-listing/black-listing should be implemented in relevant functions if possible.

For example:

Listing 10: Modifier.sol (Lines 2,3,4)

```
1  modifier validAddress(address addr) {
2      require(addr != address(0), "Address cannot be 0x0");
3      require(addr != address(this), "Address cannot be contract
           address");
4      _;
5  }
```

Remediation Plan:

SOLVED: Yieldly.Finance Team implemented address check on the related contracts.

Listing 11: Vault.sol (Lines 19)

```
18     constructor(address _multiSigAddress) Ownable() {
19         require(_multiSigAddress != address(0), "Cannot set
               address to 0");
20         multiSigAddress = _multiSigAddress;
21     }
```

3.4 (HAL-04) THRESHOLD LIMIT VALIDATION MISSING - LOW

Description:

There are multiple roles defined in the Yieldly.Finance. They are named as Signatory, Dispatcher and Validator. Each role has different functions that it can access. In the contracts, No limit has been set on the roles and these roles can have the desired number of people. The maximum value of threshold is not defined in the contracts.

Function:

Dispatcher.sol Line #~79

Listing 12: Dispatcher.sol (Lines)

```
79     function newThreshold(uint256 _threshold) public onlyMultiSig
           {
80         require(_threshold <= validators.length, "Validation
               threshold cannot exceed amount of validators");
81         require(_threshold > 0, "Threshold must be greater than 0"
               );
82         valThreshold = _threshold;
83         emit NewThresholdEvent(_threshold);
84     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to define threshold limit for validators and signatories.

Remediation Plan:

SOLVED: The maximum value of the threshold is limited by the amount of the validators and signatories. Threshold cannot be greater than the amount of signatories or validators. Therefore, Yieldly.Finance Team included their solutions according to their design.

Listing 13: Vault.sol (Lines 19)

```
80      function newThreshold(uint256 _threshold) external
            onlyMultiSig {
81          require(_threshold <= validators.length, "Validation
                threshold cannot exceed amount of validators");
82          require(_threshold > 0, "Threshold must be greater than 0"
                );
83          valThreshold = _threshold;
84          emit NewThresholdEvent(_threshold);
85      }
86
87      function newMultiSig(address _multiSigAddress) external
            onlyMultiSig {
88          require(_multiSigAddress != address(0), "Cannot set
                address to 0");
89          multiSigAddress = _multiSigAddress;
90          emit NewMultiSigEvent(_multiSigAddress);
91      }
```

3.5 (HAL-05) FLOATING PRAGMA - LOW

Description:

Yieldly.Finance contracts use the floating pragma ^0.8.0. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

Reference: ConsenSys Diligence - Lock pragmas

Code Location:

Vault.sol - Dispatcher.sol - VaultMultiSig.sol - DispatcherMultiSig.sol
Line #2-3

Listing 14: Vault.sol - Dispatcher.sol - VaultMultiSig.sol - Dispatcher-MultiSig.sol (Lines 2)

```
2 pragma solidity ^0.8.0;
```

- This is an example where the floating pragma is used. ^0.8.0.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider lock the pragma version known bugs for the compiler version. Therefore, it is recommended not to use floating pragma in the production. Apart from just locking the pragma version in the code, the sign (>=

) need to be removed. it is possible locked the pragma fixing the version both in truffle-config.js if you use the Truffle framework and in hardhat.config.js if you use HardHat framework for the deployment.

truffle-config.js

```
// Configure your compilers
compilers: {
  solc: {
     version: "0.7.5",    // Fetch exact version from solc-bin (default: truffle's version)
    // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
    // settings: {          // See the solidity docs for advice about optimization and evmVersion
    //  optimizer: {
    //    enabled: false,
    //    runs: 200
    //  },
    //  evmVersion: "byzantium"
    // }
  }
}
};
```

hardhat.config.js

```
/**
 * @type import('hardhat/config').HardhatUserConfig
 */
module.exports = {
  solidity: "0.7.5",
};
```

Remediation Plan:

Solved: Yieldly.Finance Team team considers using the pragma version 0.8.4 for some reasons:
- Using the gas improvements from 0.8.2 https://github.com/ethereum/solidity/releases/tag/v0.8.2
- A medium severity bug found on March 20th that is present with all prior versions of Solidity https://blog.soliditylang.org/2021/03/23/keccak-optimizer-bug/
- 0.8.0 and later versions include native overflow and underflow checks that used to require the SafeMath lib or custom checks to avoid. This is a welcome safety improvement that we will want in the accompanying contracts. https://blog.soliditylang.org/2020/12/16/solidity-v0.8.0-release-announcement/

Listing 15: Dispatcher.sol (Lines 1)

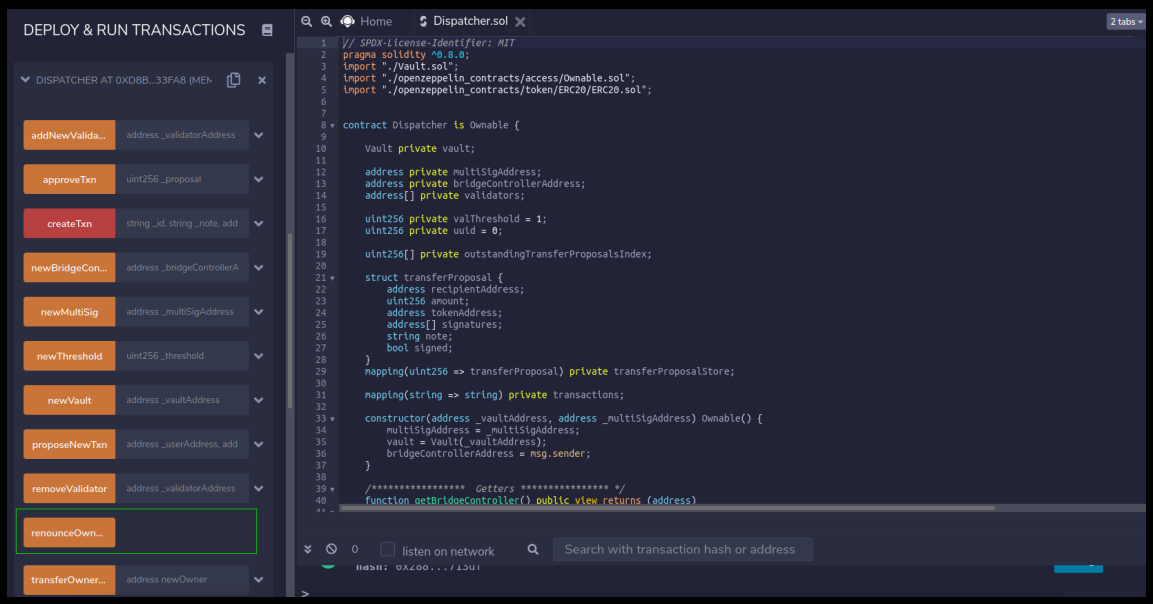
```
1 pragma solidity 0.8.4;
2 import "./Vault.sol";
3 import "./openzeppelin_contracts/access/Ownable.sol";
4 import "./openzeppelin_contracts/token/ERC20/ERC20.sol";
```

3.6 (HAL-06) OWNER CAN RENOUNCE OWNERSHIP - LOW

Description:

The Owner of the contract is usually the account which deploys the contract. As a result, the Owner is able to perform some privileged actions. In the Vault.sol, Dispatcher.sol, DispatcherMultiSig.sol and VaultMultiSig.sol smart contracts, the renounceOwnership function is used to renounce being Owner. Otherwise, if the ownership was not transferred before, the contract will never have an Owner, which is dangerous.

Function:



Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It's recommended that the Owner is not able to call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users. As an other solution, Renounce Ownership functionality can be disabled with the following line.

For example:

Listing 16: Modifier.sol (Lines 3)

```
2  function renounceOwnership() public override onlyOwner {
3          revert("can't renounceOwnership here"); //not possible
               with this smart contract
4      }
```

Remediation Plan:

SOLVED: Yieldly.Finance Team implemented the necessary changes.

Listing 17: OpenZeppelin Interface (Lines 2)

```
1      function renounceOwnership() public virtual onlyOwner {
2          revert("Cannot renounceOwnership with this contract");
3          //not possible for these contracts
4      }
```

3.7 (HAL-07) MISSING EVENT HANDLER - LOW

Description:

In the Yieldly.Finance contracts the some of functions do not emit event after the progress. Events are a method of informing the transaction initiator about the actions taken by the called function. It logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters.

Code Location:

Dispatcher.sol Line #~83

Listing 18: Dispatcher.sol (Lines)

```
79  function newThreshold(uint256 _threshold) public onlyMultiSig {
80          require(_threshold <= validators.length, "Validation
                threshold cannot exceed amount of validators");
81          require(_threshold > 0, "Threshold must be greater than 0"
                );
82          valThreshold = _threshold;
83      }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider as much as possible declaring events at the end of function. Events can be used to detect the end of the operation.

For example:

Listing 19: EmitThreshold.sol (Lines 5)

```
1  function newThreshold(uint256 _threshold) public onlyMultiSig {
2          require(_threshold <= validators.length, "Validation
               threshold cannot exceed amount of validators");
3          require(_threshold > 0, "Threshold must be greater than 0"
               );
4          valThreshold = _threshold;
5          event NewThreshold(uint256 newThreshold);
6      }
```

Remediation Plan:

SOLVED: Yieldly.Finance Team added event at the end of function.

Listing 20: Dispatcher.sol (Lines 84)

```
80     function newThreshold(uint256 _threshold) public onlyMultiSig
           {
81         require(_threshold <= validators.length, "Validation
               threshold cannot exceed amount of validators");
82         require(_threshold > 0, "Threshold must be greater than 0"
               );
83         valThreshold = _threshold;
84         emit NewThresholdEvent(_threshold);
85     }
```

3.8 (HAL-08) IGNORE RETURN VALUES - LOW

Description:

The return value of an public call is not stored in a local or state variable. In contract VaultMultiSig.sol, there are few instances where public methods are being called and return value(bool) are being ignored.

Code Location:

VaultMultiSig.sol Line #~88

Listing 21: Dispatcher.sol (Lines)

```
88      function proposeAddress(address _address, uint256 _index)
            public onlyOwner {
89          addressProposalStore[uuid].proposal = _address;
90          addressProposalStore[uuid].proposalType = _index;
91          addressProposalStore[uuid].timeStamp = block.timestamp;
92          outstandingAddressProposalsIndex.push(uuid);
93          uuid += 1;
94      }
95
96      function proposeNewOwner(address _address) public
            onlySignatories {
97          addressProposalStore[uuid].proposal = _address;
98          addressProposalStore[uuid].proposalType = 2;
99          addressProposalStore[uuid].timeStamp = block.timestamp;
100         outstandingAddressProposalsIndex.push(uuid);
101         uuid += 1;
102     }
103
104     function proposeNewThreshold(uint256 _threshold) public
            onlyOwner {
105         thresholdProposalStore[uuid].proposal = _threshold;
106         thresholdProposalStore[uuid].timeStamp = block.timestamp;
107         outstandingThresholdProposalsIndex.push(uuid);
108         uuid += 1;
109     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

Remediation Plan:

SOLVED: Yieldly.Finance Team dispatched events on the functions.

Listing 22: VaultMultiSig.sol (Lines 94)

```
88     function proposeAddress(address _address, uint256 _index)
           external onlyOwner {
89         addressProposalStore[uuid].proposal = _address;
90         addressProposalStore[uuid].proposalType = _index;
91         addressProposalStore[uuid].timeStamp = block.timestamp;
92         outstandingAddressProposalsIndex.push(uuid);
93         uuid += 1;
94         emit ProposeAddress(_address, _index);
95     }
```

Listing 23: VaultMultiSig.sol (Lines 103)

```
97     function proposeNewOwner(address _address) external
           onlySignatories {
98         addressProposalStore[uuid].proposal = _address;
99         addressProposalStore[uuid].proposalType = 2;
100        addressProposalStore[uuid].timeStamp = block.timestamp;
101        outstandingAddressProposalsIndex.push(uuid);
102        uuid += 1;
103        emit ProposeNewOwner(_address);
104    }
```

Listing 24: VaultMultiSig.sol (Lines 111)

```
106     function proposeNewThreshold(uint256 _threshold) external
            onlyOwner {
107         thresholdProposalStore[uuid].proposal = _threshold;
108         thresholdProposalStore[uuid].timeStamp = block.timestamp;
109         outstandingThresholdProposalsIndex.push(uuid);
110         uuid += 1;
111         emit ProposeNewThreshold(_threshold);
112     }
```

3.9 (HAL-09) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

Vault.sol Line #37

Listing 25: Modifier.sol (Lines 37)

```
37 function transferFunds(address _tokenAddress, address _recipient,
       uint256 _amount) public onlyDispatcher {
38         require(tokensStore[_tokenAddress].active == true, "Token
               not supported");
39         require(_amount > 0, "Cannot transfer 0 tokens");
40         ERC20(_tokenAddress).transfer(_recipient, _amount);
41         emit ReleasedFundsEvent(_recipient, _amount);
42     }
```

We noticed the use of public functions in the following contracts:

- Vault.sol

Listing 26: Vault.sol (Lines 37)

```
37 function getDispatcherAddress()
38 function getMultiSigAddress()
39 function getTokenAddresses()
40 function transferFunds()
```

```
41 function newMultiSig()
42 function newDispatcher()
43 function addToken()
44 function removeToken()
```

Dispatcher.sol

Listing 27: Dispatcher.sol (Lines 37)

```
37 function getBridgeController()
38 function getValidators()
39 function getVaultAddress()
40 function getMultiSig()
41 function getOutstandingTransferProposals()
42 function getValThreshold()
43 function getCreatedTransanction()
44 function getUUID()
45 function newThreshold()
46 function newMultiSig()
47 function newBridgeController()
48 function newVault()
49 function addNewValidator()
50 function proposeNewTxn()
51 function approveTxn()
52 function createTxn()
```

- VaultMultiSig.sol & DispatcherMultiSig.sol

Listing 28: VaultMultiSig.sol - DispatcherMultiSig.sol (Lines 37)

```
37 function getSignatories()
38 function getProposal()
39 function getThresholdProposal()
40 function getOutstandingAddressProposals()
41 function getOutstandingThresholdProposals()
42 function getThreshold()
43 function proposeAddress()
44 function proposeAddress()
45 function proposeNewOwner()
46 function proposeNewThreshold()
```

```
47 function approveSignatory()
48 function removeSignatory()
49 function approveNewOwner()
50 function approveNewMultiSig()
51 function approveNewToken()
52 function approveNewDispatcher()
53 function approveNewVault()
54 function approveNewThreshold()
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remediation Plan:

SOLVED: Yieldly.Finance Team marked functions as an external.

3.10 (HAL-10) BLOCK TIMESTAMP USAGE - INFORMATIONAL

Description:

During a manual static review, we noticed the use of block.timestamp. block.timestamp can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles.

Code Location:

Listing 29: VaultMultiSig.sol (Lines 91)

```
88     function proposeAddress(address _address, uint256 _index)
           public onlyOwner {
89         addressProposalStore[uuid].proposal = _address;
90         addressProposalStore[uuid].proposalType = _index;
91         addressProposalStore[uuid].timeStamp = block.timestamp;
92         outstandingAddressProposalsIndex.push(uuid);
93         uuid += 1;
94     }
```

We noticed the use of block.timestamp in the following contracts:
- VaultMultiSig.sol line #91,99,108
- DispatcherMultiSig.sol line #86-106

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use block.number instead of block.timestamp to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days

and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation plan:

NOT APPLICABLE: In the contracts, the timestamp is used only for the record keeping. the Yieldly.Finance Team don't need high degree of accuracy. Also, the Yieldly.Finance Team considers safe the usage of block.timestamp because 900 seconds of drift from miners is preferable to other options. Calculating time from the block could be wrong if there is a fork or upgrade - timestamps are less vulnerable to a change in block duration that could occur with Ethereum 2.0 upgrades or hard forks. Use of oracles would create a dependency on the health of a third party service and potentially incur additional fees.

3.11 (HAL-11) REDUNDANT BOOLEAN COMPARISON - INFORMATIONAL

Description:

In the solidity language, Boolean constants can be used directly and do not need to be compare to true or false. In the Yieldly.Finance contracts, boolean constants are compared with true or false.

Code Location:

Listing 30: VaultMultiSig.sol (Lines 237)

```
235  function approveNewThreshold(uint256 _proposal) public
         onlySignatories oneVoteThreshold(_proposal){
236          require(thresholdProposalStore[_proposal].signed == false,
                 "Already Signed");
237          require(thresholdProposalStore[_proposal].proposal <=
                 signatories.length, "Can't be less signatories than
                 threshold");
238          thresholdProposalStore[_proposal].signatures.push(msg.
                 sender);
239
240          if (thresholdProposalStore[_proposal].signatures.length >=
                  threshold) {
241              threshold = thresholdProposalStore[_proposal].proposal
                     ;
242              popThresholdProposal(_proposal);
243              emit ApprovedNewThreshold(thresholdProposalStore[
                     _proposal].proposal);
244          }
245      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to compare boolean constants directly in the require modifier.

Remediation plan:

SOLVED: the Yieldly.Finance Team implemented all boolean comparisions correctly. Therefore, the issue marked as a solved.

3.12 (HAL-12) FOR LOOP OVER DYNAMIC ARRAY - INFORMATIONAL

Description:

When smart contracts are deployed or functions inside them are called, the execution of these actions always requires a certain amount of gas, based on how much computation is needed to complete them. The Ethereum network specifies a block gas limit and the sum of all transactions included in a block cannot exceed the threshold.

Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition.

A situation in which the block gas limit can be an issue is in sending funds to an array of addresses. Even without any malicious intent, this can easily go wrong. Just by having too large an array of users to pay can max out the gas limit and prevent the transaction from ever succeeding.

Example Location:

Listing 31: Dispatcher.sol (Lines 112)

```
110 function removeValidator(address _validatorAddress) public
        onlyMultiSig {
111         //Remove a validator threshold count in order to avoid not
                having enough validators
112         for(uint256 i = 0; i <= validators.length; i++)
113         {
114             if(validators[i] == _validatorAddress)
115             {
116                 validators[i] = validators[validators.length - 1];
117                 validators.pop();
118                 if(valThreshold > 1)
119                 {
```

```
120                     valThreshold = valThreshold - 1;
121                 }
122                 break;
123             }
124         }
125         emit RemovedValidator(_validatorAddress);
126     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Actions that require looping across the entire data structure should be avoided. If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions.

Remediation plan:

SOLVED: The risk was taken into consideration for all looped code sections. Any contracts with loops are not accessible by public and cannot be hit with a DDOS attack. The access control mechanisms are implemented on the relevant function. Also, the Yieldly.Finance Team implemented loop breaks to avoid infinite loops.

Listing 32: Dispatcher.sol (Lines 122)

```
110 function removeValidator(address _validatorAddress) public
        onlyMultiSig {
111         //Remove a validator threshold count in order to avoid not
                having enough validators
112         for(uint256 i = 0; i <= validators.length; i++)
113         {
114             if(validators[i] == _validatorAddress)
115             {
```

```
116                 validators[i] = validators[validators.length - 1];
117                 validators.pop();
118                 if(valThreshold > 1)
119                 {
120                     valThreshold = valThreshold - 1;
121                 }
122                 break;
123             }
124         }
125         emit RemovedValidator(_validatorAddress);
126     }
```

3.13 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

Results:

Vault.sol

```
INFO:Detectors:
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) ignores return value by ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Vault.constructor(address)._multiSigAddress (contracts/Vault.sol#18) lacks a zero-check on :
        - multiSigAddress = _multiSigAddress (contracts/Vault.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42):
        External calls:
        - ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
        Event emitted after the call(s):
        - ReleasedFundsEvent(_recipient,_amount) (contracts/Vault.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported) (contracts/Vault.sol#38)
Vault.addToken(address) (contracts/Vault.sol#56-62) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active != true,Token already supported) (contracts/Vault.sol#57)
Vault.removeToken(address) (contracts/Vault.sol#64-69) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported already) (contracts/Vault.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Pragma version^0.8.0 (contracts/Vault.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Struct Vault.supportedToken (contracts/Vault.sol#11-14) is not in CapWords
Parameter Vault.transferFunds(address,address,uint256)._tokenAddress (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.transferFunds(address,address,uint256)._recipient (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.transferFunds(address,address,uint256)._amount (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.newMultiSig(address)._multiSigAddress (contracts/Vault.sol#44) is not in mixedCase
Parameter Vault.newDispatcher(address)._dispatcherAddress (contracts/Vault.sol#50) is not in mixedCase
Parameter Vault.addToken(address)._tokenAddress (contracts/Vault.sol#56) is not in mixedCase
Parameter Vault.removeToken(address)._tokenAddress (contracts/Vault.sol#64) is not in mixedCase
Parameter Vault.popTokenArray(address)._tokenAddress (contracts/Vault.sol#73) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (contracts/openzeppelin_contracts/utils/Context.sol#21)" inContext (contracts/openzeppelin_contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
INFO:Detectors:
Redundant expression "this (contracts/openzeppelin_contracts/utils/Context.sol#21)" inContext (contracts/openzeppelin_contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
getDispatcherAddress() should be declared external:
        - Vault.getDispatcherAddress() (contracts/Vault.sol#24-26)
getMultiSigAddress() should be declared external:
        - Vault.getMultiSigAddress() (contracts/Vault.sol#28-30)
getTokenAddresses() should be declared external:
        - Vault.getTokenAddresses() (contracts/Vault.sol#32-34)
transferFunds(address,address,uint256) should be declared external:
        - Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42)
newMultiSig(address) should be declared external:
        - Vault.newMultiSig(address) (contracts/Vault.sol#44-48)
newDispatcher(address) should be declared external:
        - Vault.newDispatcher(address) (contracts/Vault.sol#50-54)
addToken(address) should be declared external:
        - Vault.addToken(address) (contracts/Vault.sol#56-62)
removeToken(address) should be declared external:
        - Vault.removeToken(address) (contracts/Vault.sol#64-69)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/openzeppelin_contracts/access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/openzeppelin_contracts/access/Ownable.sol#63-67)
name() should be declared external:
        - ERC20.name() (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#59-61)
symbol() should be declared external:
        - ERC20.symbol() (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#67-69)
decimals() should be declared external:
        - ERC20.decimals() (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#84-86)
totalSupply() should be declared external:
        - ERC20.totalSupply() (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#91-93)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#98-100)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#110-113)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#118-120)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#129-132)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#147-155)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#169-172)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#188-194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/Vault.sol analyzed (5 contracts with 72 detectors), 43 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Dispatcher.sol

```
INFO:Detectors:
Reentrancy in Dispatcher.proposeNewTxn(address,address,uint256,string) (contracts/Dispatcher.sol#131-150):
        External calls:
        - vault.transferFunds(transferProposalStore[uuid].tokenAddress,transferProposalStore[uuid].recipientAddress,transferProposalStore[uuid].amount) (contracts/Dispatcher.sol#138)
        State variables written after the call(s):
        - transferProposalStore[uuid].signed = true (contracts/Dispatcher.sol#141)
        - uuid += 1 (contracts/Dispatcher.sol#149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178) ignores return value by ERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amount) (c
ontracts/Dispatcher.sol#175)
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) ignores return value by ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Dispatcher.newThreshold(uint256) (contracts/Dispatcher.sol#79-83) should emit an event for:
        - valThreshold = _threshold (contracts/Dispatcher.sol#82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Dispatcher.constructor(address,address)._multiSigAddress (contracts/Dispatcher.sol#33) lacks a zero-check on :
                - multiSigAddress = _multiSigAddress (contracts/Dispatcher.sol#34)
Vault.constructor(address)._multiSigAddress (contracts/Vault.sol#18) lacks a zero-check on :
                - multiSigAddress = _multiSigAddress (contracts/Vault.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Modifier Dispatcher.onlyValidators() (contracts/Dispatcher.sol#206-212) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrec
t-modifier
INFO:Detectors:
Reentrancy in Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162):
        External calls:
        - vault.transferFunds(transferProposalStore[_proposal].tokenAddress,transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount) (contracts/Dispatcher.
sol#158)
        State variables written after the call(s):
        - popTransferProposal(_proposal) (contracts/Dispatcher.sol#159)
                - outstandingTransferProposalsIndex[L] = outstandingTransferProposalsIndex[outstandingTransferProposalsIndex.length - 1] (contracts/Dispatcher.sol#188)
                - outstandingTransferProposalsIndex.pop() (contracts/Dispatcher.sol#189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162):
        External calls:
        - vault.transferFunds(transferProposalStore[_proposal].tokenAddress,transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount) (contracts/Dispatcher.
sol#158)
        Event emitted after the call(s):
        - ApprovedTransaction(transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount,_proposal) (contracts/Dispatcher.sol#160)
Reentrancy in Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178):
        External calls:
        - ERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amount) (contracts/Dispatcher.sol#175)
        External calls sending eth:
        - address(bridgeControllerAddress).transfer(msg.value) (contracts/Dispatcher.sol#176)
        Event emitted after the call(s):
        - NewTransactionCreated(msg.sender,_tokenAddress,_amount) (contracts/Dispatcher.sol#177)
```

```
Reentrancy in Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162):
        External calls:
        - vault.transferFunds(transferProposalStore[_proposal].tokenAddress,transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount) (contracts/Dispatcher
sol#158)
        Event emitted after the call(s):
        - ApprovedTransaction(transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount,_proposal) (contracts/Dispatcher.sol#160)
Reentrancy in Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178):
        External calls:
        - ERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amount) (contracts/Dispatcher.sol#175)
        External calls sending eth:
        - address(bridgeControllerAddress).transfer(msg.value) (contracts/Dispatcher.sol#176)
        Event emitted after the call(s):
        - NewTransactionCreated(msg.sender,_tokenAddress,_amount) (contracts/Dispatcher.sol#177)
Reentrancy in Dispatcher.proposeNewTxn(address,address,uint256,string) (contracts/Dispatcher.sol#131-150):
        External calls:
        - vault.transferFunds(transferProposalStore[uuid].tokenAddress,transferProposalStore[uuid].reciplentAddress,transferProposalStore[uuid].amount) (contracts/Dispatcher.sol#138)
        Event emitted after the call(s):
        - ApprovedTransaction(transferProposalStore[uuid].recipientAddress,transferProposalStore[uuid].amount,uuid) (contracts/Dispatcher.sol#139)
        - proposalCreated(uuid) (contracts/Dispatcher.sol#140)
Reentrancy in Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42):
        External calls:
        - ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
        Event emitted after the call(s):
        - ReleasedFundsEvent(_recipient,_amount) (contracts/Vault.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162) compares to a boolean constant:
        -require(bool,string)(transferProposalStore[_proposal].signed == false,Already Signed) (contracts/Dispatcher.sol#153)
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported) (contracts/Vault.sol#38)
Vault.addToken(address) (contracts/Vault.sol#56-62) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active != true,Token already supported) (contracts/Vault.sol#57)
Vault.removeToken(address) (contracts/Vault.sol#64-69) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported already) (contracts/Vault.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Dispatcher.removeValidator(address) (contracts/Dispatcher.sol#110-126) has costly operations inside a loop:
        - valThreshold = valThreshold - 1 (contracts/Dispatcher.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Pragma version^0.8.0 (contracts/Dispatcher.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/Vault.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
getBridgeController() should be declared external:
        - Dispatcher.getBridgeController() (contracts/Dispatcher.sol#40-43)
getValidators() should be declared external:
        - Dispatcher.getValidators() (contracts/Dispatcher.sol#45-48)
getVaultAddress() should be declared external:
        - Dispatcher.getVaultAddress() (contracts/Dispatcher.sol#50-53)
getMultiSig() should be declared external:
        - Dispatcher.getMultiSig() (contracts/Dispatcher.sol#55-58)
getOutstandingTransferProposals() should be declared external:
        - Dispatcher.getOutstandingTransferProposals() (contracts/Dispatcher.sol#60-62)
getValThreshold() should be declared external:
        - Dispatcher.getValThreshold() (contracts/Dispatcher.sol#64-67)
getCreatedTransanction(string) should be declared external:
        - Dispatcher.getCreatedTransanction(string) (contracts/Dispatcher.sol#68-71)
getUUID() should be declared external:
        - Dispatcher.getUUID() (contracts/Dispatcher.sol#73-76)
newThreshold(uint256) should be declared external:
        - Dispatcher.newThreshold(uint256) (contracts/Dispatcher.sol#79-83)
newMultiSig(address) should be declared external:
        - Dispatcher.newMultiSig(address) (contracts/Dispatcher.sol#85-89)
newVault(address) should be declared external:
        - Dispatcher.newVault(address) (contracts/Dispatcher.sol#92-96)
newBridgeController(address) should be declared external:
        - Dispatcher.newBridgeController(address) (contracts/Dispatcher.sol#98-102)
addNewValidator(address) should be declared external:
        - Dispatcher.addNewValidator(address) (contracts/Dispatcher.sol#104-108)
removeValidator(address) should be declared external:
        - Dispatcher.removeValidator(address) (contracts/Dispatcher.sol#110-126)
proposeNewTxn(address,address,uint256,string) should be declared external:
        - Dispatcher.proposeNewTxn(address,address,uint256,string) (contracts/Dispatcher.sol#131-150)
approveTxn(uint256) should be declared external:
        - Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162)
createTxn(string,string,address,uint256,uint256) should be declared external:
        - Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178)
getDispatcherAddress() should be declared external:
        - Vault.getDispatcherAddress() (contracts/Vault.sol#24-26)
getMultiSigAddress() should be declared external:
        - Vault.getMultiSigAddress() (contracts/Vault.sol#28-30)
getTokenAddresses() should be declared external:
        - Vault.getTokenAddresses() (contracts/Vault.sol#32-34)
transferFunds(address,address,uint256) should be declared external:
        - Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42)
newMultiSig(address) should be declared external:
        - Vault.newMultiSig(address) (contracts/Vault.sol#44-48)
newDispatcher(address) should be declared external:
        - Vault.newDispatcher(address) (contracts/Vault.sol#50-54)
addToken(address) should be declared external:
        - Vault.addToken(address) (contracts/Vault.sol#56-62)
removeToken(address) should be declared external:
        - Vault.removeToken(address) (contracts/Vault.sol#64-69)
```

VaultMultiSig.sol

```
INFO:Detectors:
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) ignores return value by ERC20(_tokenAddress).transfer(_reciipient,_amount) (contracts/Vault.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Vault.constructor(address)._multiSigAddress (contracts/Vault.sol#18) lacks a zero-check on :
        - multiSigAddress = _multiSigAddress (contracts/Vault.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Modifier VaultMultiSig.onlySignatories() (contracts/VaultMultiSig.sol#286-292) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
INFO:Detectors:
Reentrancy in VaultMultiSig.approveNewDispatcher(uint256) (contracts/VaultMultiSig.sol#205-218):
        External calls:
        - vault.newDispatcher(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#214)
        State variables written after the call(s):
        - popAddressProposal(_proposal) (contracts/VaultMultiSig.sol#215)
                - outstandingAddressProposalsIndex[i] = outstandingAddressProposalsIndex[outstandingAddressProposalsIndex.length - 1] (contracts/VaultMultiSig.sol#254)
                - outstandingAddressProposalsIndex.pop() (contracts/VaultMultiSig.sol#255)
Reentrancy in VaultMultiSig.approveNewMultiSig(uint256) (contracts/VaultMultiSig.sol#160-173):
        External calls:
        - vault.newMultiSig(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#169)
        State variables written after the call(s):
        - popAddressProposal(_proposal) (contracts/VaultMultiSig.sol#170)
                - outstandingAddressProposalsIndex[i] = outstandingAddressProposalsIndex[outstandingAddressProposalsIndex.length - 1] (contracts/VaultMultiSig.sol#254)
                - outstandingAddressProposalsIndex.pop() (contracts/VaultMultiSig.sol#255)
Reentrancy in VaultMultiSig.approveNewToken(uint256) (contracts/VaultMultiSig.sol#175-188):
        External calls:
        - vault.addToken(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#184)
        State variables written after the call(s):
        - popAddressProposal(_proposal) (contracts/VaultMultiSig.sol#185)
                - outstandingAddressProposalsIndex[i] = outstandingAddressProposalsIndex[outstandingAddressProposalsIndex.length - 1] (contracts/VaultMultiSig.sol#254)
                - outstandingAddressProposalsIndex.pop() (contracts/VaultMultiSig.sol#255)
Reentrancy in VaultMultiSig.removeToken(uint256) (contracts/VaultMultiSig.sol#190-203):
        External calls:
        - vault.removeToken(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#199)
        State variables written after the call(s):
        - popAddressProposal(_proposal) (contracts/VaultMultiSig.sol#200)
                - outstandingAddressProposalsIndex[i] = outstandingAddressProposalsIndex[outstandingAddressProposalsIndex.length - 1] (contracts/VaultMultiSig.sol#254)
                - outstandingAddressProposalsIndex.pop() (contracts/VaultMultiSig.sol#255)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Reentrancy in VaultMultiSig.approveNewDispatcher(uint256) (contracts/VaultMultiSig.sol#205-218):
        External calls:
        - vault.newDispatcher(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#214)
        Event emitted after the call(s):
        - ApprovedDispatcher(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#216)
Reentrancy in VaultMultiSig.approveNewMultiSig(uint256) (contracts/VaultMultiSig.sol#160-173):
        External calls:
        - vault.newMultiSig(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#169)
        Event emitted after the call(s):
        - ApprovedNewMultiSig(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#171)
Reentrancy in VaultMultiSig.approveNewToken(uint256) (contracts/VaultMultiSig.sol#175-188):
        External calls:
        - vault.addToken(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#184)
        Event emitted after the call(s):
        - ApprovedNewToken(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#186)
Reentrancy in VaultMultiSig.removeToken(uint256) (contracts/VaultMultiSig.sol#190-203):
        External calls:
        - vault.removeToken(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#199)
        Event emitted after the call(s):
        - RemovedToken(addressProposalStore[_proposal].proposal) (contracts/VaultMultiSig.sol#201)
Reentrancy in Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42):
        External calls:
        - ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
        Event emitted after the call(s):
        - ReleasedFundsEvent(_recipient,_amount) (contracts/Vault.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported) (contracts/Vault.sol#38)
Vault.addToken(address) (contracts/Vault.sol#56-62) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active != true,Token already supported) (contracts/Vault.sol#57)
Vault.removeToken(address) (contracts/Vault.sol#64-69) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported already) (contracts/Vault.sol#65)
VaultMultiSig.approveSignatory(uint256) (contracts/VaultMultiSig.sol#112-124) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#114)
VaultMultiSig.removeSignatory(uint256) (contracts/VaultMultiSig.sol#126-143) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#128)
VaultMultiSig.approveNewOwner(uint256) (contracts/VaultMultiSig.sol#145-158) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#147)
VaultMultiSig.approveNewMultiSig(uint256) (contracts/VaultMultiSig.sol#160-173) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#162)
VaultMultiSig.approveNewToken(uint256) (contracts/VaultMultiSig.sol#175-188) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#177)
VaultMultiSig.removeToken(uint256) (contracts/VaultMultiSig.sol#190-203) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#192)
VaultMultiSig.approveNewDispatcher(uint256) (contracts/VaultMultiSig.sol#205-218) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#207)
VaultMultiSig.approveNewVault(uint256) (contracts/VaultMultiSig.sol#220-233) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#222)
VaultMultiSig.approveNewThreshold(uint256) (contracts/VaultMultiSig.sol#235-245) compares to a boolean constant:
        -require(bool,string)(thresholdProposalStore[_proposal].signed == false,Already Signed) (contracts/VaultMultiSig.sol#236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
INFO:Detectors:
Struct Vault.supportedToken (contracts/Vault.sol#11-14) is not in CapWords
Parameter Vault.transferFunds(address,address,uint256)._tokenAddress (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.transferFunds(address,address,uint256)._recipient (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.transferFunds(address,address,uint256)._amount (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.newMultiSig(address)._multiSigAddress (contracts/Vault.sol#44) is not in mixedCase
Parameter Vault.newDispatcher(address)._dispatcherAddress (contracts/Vault.sol#50) is not in mixedCase
Parameter Vault.addToken(address)._tokenAddress (contracts/Vault.sol#56) is not in mixedCase
Parameter Vault.removeToken(address)._tokenAddress (contracts/Vault.sol#64) is not in mixedCase
Parameter Vault.popTokenArray(address)._tokenAddress (contracts/Vault.sol#73) is not in mixedCase
Struct VaultMultiSig.addressProposal (contracts/VaultMultiSig.sol#28-34) is not in CapWords
Struct VaultMultiSig.thresholdProposal (contracts/VaultMultiSig.sol#37-42) is not in CapWords
Parameter VaultMultiSig.getProposal(uint256)._index (contracts/VaultMultiSig.sol#54) is not in mixedCase
Parameter VaultMultiSig.getThresholdProposal(uint256)._index (contracts/VaultMultiSig.sol#58) is not in mixedCase
Parameter VaultMultiSig.proposeAddress(address,uint256)._address (contracts/VaultMultiSig.sol#88) is not in mixedCase
Parameter VaultMultiSig.proposeAddress(address,uint256)._index (contracts/VaultMultiSig.sol#88) is not in mixedCase
Parameter VaultMultiSig.proposeNewOwner(address)._address (contracts/VaultMultiSig.sol#96) is not in mixedCase
Parameter VaultMultiSig.proposeNewThreshold(uint256)._threshold (contracts/VaultMultiSig.sol#104) is not in mixedCase
Parameter VaultMultiSig.approveSignatory(uint256)._proposal (contracts/VaultMultiSig.sol#112) is not in mixedCase
Parameter VaultMultiSig.removeSignatory(uint256)._proposal (contracts/VaultMultiSig.sol#126) is not in mixedCase
Parameter VaultMultiSig.approveNewOwner(uint256)._proposal (contracts/VaultMultiSig.sol#145) is not in mixedCase
Parameter VaultMultiSig.approveNewMultiSig(uint256)._proposal (contracts/VaultMultiSig.sol#160) is not in mixedCase
Parameter VaultMultiSig.approveNewToken(uint256)._proposal (contracts/VaultMultiSig.sol#175) is not in mixedCase
Parameter VaultMultiSig.removeToken(uint256)._proposal (contracts/VaultMultiSig.sol#190) is not in mixedCase
Parameter VaultMultiSig.approveNewDispatcher(uint256)._proposal (contracts/VaultMultiSig.sol#205) is not in mixedCase
Parameter VaultMultiSig.approveNewVault(uint256)._proposal (contracts/VaultMultiSig.sol#220) is not in mixedCase
Parameter VaultMultiSig.approveNewThreshold(uint256)._proposal (contracts/VaultMultiSig.sol#235) is not in mixedCase
Parameter VaultMultiSig.popAddressProposal(uint256)._uuid (contracts/VaultMultiSig.sol#249) is not in mixedCase
Parameter VaultMultiSig.popThresholdProposal(uint256)._uuid (contracts/VaultMultiSig.sol#261) is not in mixedCase
Parameter VaultMultiSig.removeSignatory(address)._signatory (contracts/VaultMultiSig.sol#273) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (contracts/openzeppelin_contracts/utils/Context.sol#21)" inContext (contracts/openzeppelin_contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

DispatcherMultiSig.sol

```
INFO:Detectors:
Reentrancy in Dispatcher.proposeNewTxn(address,address,uint256,string) (contracts/Dispatcher.sol#131-150):
        External calls:
        - vault.transferFunds(transferProposalStore[uuid].tokenAddress,transferProposalStore[uuid].recipientAddress,transferProposalStore[uuid].amount) (contracts/Dispatcher.sol#138)
        State variables written after the call(s):
        - transferProposalStore[uuid].signed = true (contracts/Dispatcher.sol#141)
        - uuid += 1 (contracts/Dispatcher.sol#149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178) ignores return value by ERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amount) (c
ontracts/Dispatcher.sol#175)
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) ignores return value by ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Dispatcher.newThreshold(uint256) (contracts/Dispatcher.sol#79-83) should emit an event for:
        - valThreshold = _threshold (contracts/Dispatcher.sol#82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Dispatcher.constructor(address,address)._multiSigAddress (contracts/Dispatcher.sol#33) lacks a zero-check on :
                - multiSigAddress = _multiSigAddress (contracts/Dispatcher.sol#34)
Vault.constructor(address)._multiSigAddress (contracts/Vault.sol#18) lacks a zero-check on :
                - multiSigAddress = _multiSigAddress (contracts/Vault.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Modifier Dispatcher.onlyValidators() (contracts/Dispatcher.sol#206-212) does not always execute _; or revertModifier DispatcherMultiSig.onlySignatories() (contracts/DispatcherMultiSig.sol#2
54-260) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
INFO:Detectors:
Reentrancy in DispatcherMultiSig.approveNewMultiSig(uint256) (contracts/DispatcherMultiSig.sol#158-171):
        External calls:
        - dispatcher.newMultiSig(addressProposalStore[_proposal].proposal) (contracts/DispatcherMultiSig.sol#167)
        State variables written after the call(s):
        - popAddressProposal(_proposal) (contracts/DispatcherMultiSig.sol#168)
                - outstandingAddressProposalsIndex[i] = outstandingAddressProposalsIndex[outstandingAddressProposalsIndex.length - 1] (contracts/DispatcherMultiSig.sol#222)
                - outstandingAddressProposalsIndex.pop() (contracts/DispatcherMultiSig.sol#223)
Reentrancy in DispatcherMultiSig.approveNewVault(uint256) (contracts/DispatcherMultiSig.sol#173-186):
        External calls:
        - dispatcher.newVault(addressProposalStore[_proposal].proposal) (contracts/DispatcherMultiSig.sol#182)
        State variables written after the call(s):
        - popAddressProposal(_proposal) (contracts/DispatcherMultiSig.sol#183)
                - outstandingAddressProposalsIndex[i] = outstandingAddressProposalsIndex[outstandingAddressProposalsIndex.length - 1] (contracts/DispatcherMultiSig.sol#222)
                - outstandingAddressProposalsIndex.pop() (contracts/DispatcherMultiSig.sol#223)
Reentrancy in Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162):
        External calls:
        - vault.transferFunds(transferProposalStore[_proposal].tokenAddress,transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount) (contracts/Dispatcher.
sol#158)
```

```
INFO:Detectors:
Reentrancy in DispatcherMultiSig.approveNewMultiSig(uint256) (contracts/DispatcherMultiSig.sol#158-171):
        External calls:
        - dispatcher.newMultiSig(addressProposalStore[_proposal].proposal) (contracts/DispatcherMultiSig.sol#167)
        Event emitted after the call(s):
        - ApprovedNewMultiSig(addressProposalStore[_proposal].proposal) (contracts/DispatcherMultiSig.sol#169)
Reentrancy in DispatcherMultiSig.approveNewVault(uint256) (contracts/DispatcherMultiSig.sol#173-186):
        External calls:
        - dispatcher.newVault(addressProposalStore[_proposal].proposal) (contracts/DispatcherMultiSig.sol#182)
        Event emitted after the call(s):
        - ApprovedNewVault(addressProposalStore[_proposal].proposal) (contracts/DispatcherMultiSig.sol#184)
Reentrancy in Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162):
        External calls:
        - vault.transferFunds(transferProposalStore[_proposal].tokenAddress,transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount) (contracts/Dispatcher.
sol#158)
        Event emitted after the call(s):
        - ApprovedTransaction(transferProposalStore[_proposal].recipientAddress,transferProposalStore[_proposal].amount,_proposal) (contracts/Dispatcher.sol#160)
Reentrancy in Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178):
        External calls:
        - ERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amount) (contracts/Dispatcher.sol#175)
        External calls sending eth:
        - address(bridgeControllerAddress).transfer(msg.value) (contracts/Dispatcher.sol#176)
        Event emitted after the call(s):
        - NewTransactionCreated(msg.sender,_tokenAddress,_amount) (contracts/Dispatcher.sol#177)
Reentrancy in Dispatcher.proposeNewTxn(address,address,uint256,string) (contracts/Dispatcher.sol#131-150):
        External calls:
        - vault.transferFunds(transferProposalStore[uuid].tokenAddress,transferProposalStore[uuid].recipientAddress,transferProposalStore[uuid].amount) (contracts/Dispatcher.sol#138)
        Event emitted after the call(s):
        - ApprovedTransaction(transferProposalStore[uuid].recipientAddress,transferProposalStore[uuid].amount,uuid) (contracts/Dispatcher.sol#139)
        - proposalCreated(uuid) (contracts/Dispatcher.sol#140)
Reentrancy in Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42):
        External calls:
        - ERC20(_tokenAddress).transfer(_recipient,_amount) (contracts/Vault.sol#40)
        Event emitted after the call(s):
        - ReleasedFundsEvent(_recipient,_amount) (contracts/Vault.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162) compares to a boolean constant:
        -require(bool,string)(transferProposalStore[_proposal].signed == false,Already Signed) (contracts/Dispatcher.sol#153)
DispatcherMultiSig.approveSignatory(uint256) (contracts/DispatcherMultiSig.sol#110-122) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#112)
DispatcherMultiSig.removeSignatory(uint256) (contracts/DispatcherMultiSig.sol#124-141) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#126)
DispatcherMultiSig.approveNewOwner(uint256) (contracts/DispatcherMultiSig.sol#143-156) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#145)
DispatcherMultiSig.approveNewMultiSig(uint256) (contracts/DispatcherMultiSig.sol#158-171) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#160)
DispatcherMultiSig.approveNewVault(uint256) (contracts/DispatcherMultiSig.sol#173-186) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#175)
DispatcherMultiSig.approveNewDispatcher(uint256) (contracts/DispatcherMultiSig.sol#188-201) compares to a boolean constant:
        -require(bool,string)(addressProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#190)
DispatcherMultiSig.approveNewThreshold(uint256) (contracts/DispatcherMultiSig.sol#203-213) compares to a boolean constant:
        -require(bool,string)(thresholdProposalStore[_proposal].signed == false,Already Signed) (contracts/DispatcherMultiSig.sol#204)
Vault.transferFunds(address,address,uint256) (contracts/Vault.sol#37-42) compares to a boolean constant:
        -require(bool,string)(tokensStore[_tokenAddress].active == true,Token not supported) (contracts/Vault.sol#38)
```

```
INFO:Detectors:
Dispatcher.removeValidator(address) (contracts/Dispatcher.sol#110-126) has costly operations inside a loop:
        - valThreshold = valThreshold - 1 (contracts/Dispatcher.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Pragma version^0.8.0 (contracts/Dispatcher.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/DispatcherMultiSig.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/Vault.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/token/ERC20/ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (contracts/openzeppelin_contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Struct Dispatcher.transferProposal (contracts/Dispatcher.sol#21-28) is not in CapWords
Event DispatcherproposalCreated(uint256) (contracts/Dispatcher.sol#229) is not in CapWords
Parameter Dispatcher.newThreshold(uint256)._threshold (contracts/Dispatcher.sol#79) is not in mixedCase
Parameter Dispatcher.newMultiSig(address)._multiSigAddress (contracts/Dispatcher.sol#85) is not in mixedCase
Parameter Dispatcher.newVault(address)._vaultAddress (contracts/Dispatcher.sol#92) is not in mixedCase
Parameter Dispatcher.newBridgeController(address)._bridgeControllerAddress (contracts/Dispatcher.sol#98) is not in mixedCase
Parameter Dispatcher.addNewValidator(address)._validatorAddress (contracts/Dispatcher.sol#104) is not in mixedCase
Parameter Dispatcher.removeValidator(address)._validatorAddress (contracts/Dispatcher.sol#110) is not in mixedCase
Parameter Dispatcher.proposeNewTxn(address,address,uint256,string)._userAddress (contracts/Dispatcher.sol#131) is not in mixedCase
Parameter Dispatcher.proposeNewTxn(address,address,uint256,string)._tokenAddress (contracts/Dispatcher.sol#131) is not in mixedCase
Parameter Dispatcher.proposeNewTxn(address,address,uint256,string)._amount (contracts/Dispatcher.sol#131) is not in mixedCase
Parameter Dispatcher.proposeNewTxn(address,address,uint256,string)._note (contracts/Dispatcher.sol#131) is not in mixedCase
Parameter Dispatcher.approveTxn(uint256)._proposal (contracts/Dispatcher.sol#152) is not in mixedCase
Parameter Dispatcher.createTxn(string,string,address,uint256,uint256)._id (contracts/Dispatcher.sol#165) is not in mixedCase
Parameter Dispatcher.createTxn(string,string,address,uint256,uint256)._note (contracts/Dispatcher.sol#166) is not in mixedCase
Parameter Dispatcher.createTxn(string,string,address,uint256,uint256)._tokenAddress (contracts/Dispatcher.sol#167) is not in mixedCase
Parameter Dispatcher.createTxn(string,string,address,uint256,uint256)._calculatedFee (contracts/Dispatcher.sol#168) is not in mixedCase
Parameter Dispatcher.createTxn(string,string,address,uint256,uint256)._amount (contracts/Dispatcher.sol#169) is not in mixedCase
Parameter Dispatcher.popTransferProposal(uint256)._uuid (contracts/Dispatcher.sol#183) is not in mixedCase
Struct DispatcherMultiSig.addressProposal (contracts/DispatcherMultiSig.sol#26-32) is not in CapWords
Struct DispatcherMultiSig.thresholdProposal (contracts/DispatcherMultiSig.sol#35-40) is not in CapWords
Parameter DispatcherMultiSig.getProposal(uint256)._index (contracts/DispatcherMultiSig.sol#52) is not in mixedCase
Parameter DispatcherMultiSig.getThresholdProposal(uint256)._index (contracts/DispatcherMultiSig.sol#56) is not in mixedCase
Parameter DispatcherMultiSig.proposeAddress(address,uint256)._address (contracts/DispatcherMultiSig.sol#86) is not in mixedCase
Parameter DispatcherMultiSig.proposeAddress(address,uint256)._index (contracts/DispatcherMultiSig.sol#86) is not in mixedCase
Parameter DispatcherMultiSig.proposeNewOwner(address)._address (contracts/DispatcherMultiSig.sol#94) is not in mixedCase
Parameter DispatcherMultiSig.proposeNewThreshold(uint256)._threshold (contracts/DispatcherMultiSig.sol#102) is not in mixedCase
Parameter DispatcherMultiSig.approveSignatory(uint256)._proposal (contracts/DispatcherMultiSig.sol#110) is not in mixedCase
Parameter DispatcherMultiSig.removeSignatory(uint256)._proposal (contracts/DispatcherMultiSig.sol#124) is not in mixedCase
Parameter DispatcherMultiSig.approveNewOwner(uint256)._proposal (contracts/DispatcherMultiSig.sol#143) is not in mixedCase
Parameter DispatcherMultiSig.approveNewMultiSig(uint256)._proposal (contracts/DispatcherMultiSig.sol#158) is not in mixedCase
Parameter DispatcherMultiSig.approveNewVault(uint256)._proposal (contracts/DispatcherMultiSig.sol#173) is not in mixedCase
Parameter DispatcherMultiSig.approveNewDispatcher(uint256)._proposal (contracts/DispatcherMultiSig.sol#188) is not in mixedCase
Parameter DispatcherMultiSig.approveNewThreshold(uint256)._proposal (contracts/DispatcherMultiSig.sol#203) is not in mixedCase
Parameter DispatcherMultiSig.popAddressProposal(uint256)._uuid (contracts/DispatcherMultiSig.sol#217) is not in mixedCase
Parameter DispatcherMultiSig.popThresholdProposal(uint256)._uuid (contracts/DispatcherMultiSig.sol#229) is not in mixedCase
Parameter DispatcherMultiSig.removeSignatory(address)._signatory (contracts/DispatcherMultiSig.sol#241) is not in mixedCase
Struct Vault.supportedToken (contracts/Vault.sol#11-14) is not in CapWords
Parameter Vault.transferFunds(address,address,uint256)._tokenAddress (contracts/Vault.sol#37) is not in mixedCase
Parameter Vault.transferFunds(address,address,uint256)._recipient (contracts/Vault.sol#37) is not in mixedCase
```

```
INFO:Detectors:
Redundant expression "this (contracts/openzeppelin_contracts/utils/Context.sol#21)" inContext (contracts/openzeppelin_contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Reentrancy in Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178):
        External calls:
        - address(bridgeControllerAddress).transfer(msg.value) (contracts/Dispatcher.sol#176)
        Event emitted after the call(s):
        - NewTransactionCreated(msg.sender,_tokenAddress,_amount) (contracts/Dispatcher.sol#177)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
getBridgeController() should be declared external:
        - Dispatcher.getBridgeController() (contracts/Dispatcher.sol#40-43)
getValidators() should be declared external:
        - Dispatcher.getValidators() (contracts/Dispatcher.sol#45-48)
getVaultAddress() should be declared external:
        - Dispatcher.getVaultAddress() (contracts/Dispatcher.sol#50-53)
getMultiSig() should be declared external:
        - Dispatcher.getMultiSig() (contracts/Dispatcher.sol#55-58)
getOutstandingTransferProposals() should be declared external:
        - Dispatcher.getOutstandingTransferProposals() (contracts/Dispatcher.sol#60-62)
getValThreshold() should be declared external:
        - Dispatcher.getValThreshold() (contracts/Dispatcher.sol#64-67)
getCreatedTransanction(string) should be declared external:
        - Dispatcher.getCreatedTransanction(string) (contracts/Dispatcher.sol#68-71)
getUUID() should be declared external:
        - Dispatcher.getUUID() (contracts/Dispatcher.sol#73-76)
newThreshold(uint256) should be declared external:
        - Dispatcher.newThreshold(uint256) (contracts/Dispatcher.sol#79-83)
newMultiSig(address) should be declared external:
        - Dispatcher.newMultiSig(address) (contracts/Dispatcher.sol#85-89)
newVault(address) should be declared external:
        - Dispatcher.newVault(address) (contracts/Dispatcher.sol#92-96)
newBridgeController(address) should be declared external:
        - Dispatcher.newBridgeController(address) (contracts/Dispatcher.sol#98-102)
addNewValidator(address) should be declared external:
        - Dispatcher.addNewValidator(address) (contracts/Dispatcher.sol#104-108)
removeValidator(address) should be declared external:
        - Dispatcher.removeValidator(address) (contracts/Dispatcher.sol#110-126)
proposeNewTxn(address,address,uint256,string) should be declared external:
        - Dispatcher.proposeNewTxn(address,address,uint256,string) (contracts/Dispatcher.sol#131-150)
approveTxn(uint256) should be declared external:
        - Dispatcher.approveTxn(uint256) (contracts/Dispatcher.sol#152-162)
createTxn(string,string,address,uint256,uint256) should be declared external:
        - Dispatcher.createTxn(string,string,address,uint256,uint256) (contracts/Dispatcher.sol#164-178)
getSignatories() should be declared external:
        - DispatcherMultiSig.getSignatories() (contracts/DispatcherMultiSig.sol#48-50)
getProposal(uint256) should be declared external:
        - DispatcherMultiSig.getProposal(uint256) (contracts/DispatcherMultiSig.sol#52-54)
getThresholdProposal(uint256) should be declared external:
        - DispatcherMultiSig.getThresholdProposal(uint256) (contracts/DispatcherMultiSig.sol#56-59)
getOutstandingAddressProposals() should be declared external:
        - DispatcherMultiSig.getOutstandingAddressProposals() (contracts/DispatcherMultiSig.sol#61-63)
```

According to the test results, most of the findings found by slither were considered as false positives. Relevant findings were reviewed by the auditors.

3.14 AUTOMATED SECURITY SCAN RESULTS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. In addition, security detections are only in scope.

Results:

Vault.sol

Report for Vault.sol
https://dashboard.mythx.io/#/console/analyses/294e892b-e05f-4ddb-8004-719906a49dc6

Line	SWC Title	Severity	Short Description
24	(SWC-000) Unknown	Medium	Function could be marked as external.
28	(SWC-000) Unknown	Medium	Function could be marked as external.
32	(SWC-000) Unknown	Medium	Function could be marked as external.

Dispatcher.sol

Report for Dispatcher.sol
https://dashboard.mythx.io/#/console/analyses/294e892b-e05f-4ddb-8004-719906a49dc6

Line	SWC Title	Severity	Short Description
40	(SWC-000) Unknown	Medium	Function could be marked as external.
45	(SWC-000) Unknown	Medium	Function could be marked as external.
50	(SWC-000) Unknown	Medium	Function could be marked as external.
55	(SWC-000) Unknown	Medium	Function could be marked as external.
60	(SWC-000) Unknown	Medium	Function could be marked as external.
64	(SWC-000) Unknown	Medium	Function could be marked as external.
68	(SWC-000) Unknown	Medium	Function could be marked as external.
73	(SWC-000) Unknown	Medium	Function could be marked as external.
164	(SWC-000) Unknown	Medium	Function could be marked as external.
176	(SWC-134) Message call with hardcoded gas amount	Low	Call with hardcoded gas amount.

VaultMultiSig.sol

Report for VaultMultiSig.sol
https://dashboard.mythx.io/#/console/analyses/c2cd108c-738e-40b5-b6c4-a73a55f364c8

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
45	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
50	(SWC-000) Unknown	Medium	Function could be marked as external.
54	(SWC-000) Unknown	Medium	Function could be marked as external.
58	(SWC-000) Unknown	Medium	Function could be marked as external.
63	(SWC-000) Unknown	Medium	Function could be marked as external.
67	(SWC-000) Unknown	Medium	Function could be marked as external.
71	(SWC-000) Unknown	Medium	Function could be marked as external.
75	(SWC-000) Unknown	Medium	Function could be marked as external.
83	(SWC-000) Unknown	Medium	Function could be marked as external.

DispatcherMultiSig.sol

Report for DispatcherMultiSig.sol
https://dashboard.mythx.io/#/console/analyses/294e892b-e05f-4ddb-8004-719906a49dc6

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
43	(SWC-100) Function Default Visibility	Low	Function visibility is not set.
48	(SWC-000) Unknown	Medium	Function could be marked as external.
52	(SWC-000) Unknown	Medium	Function could be marked as external.
56	(SWC-000) Unknown	Medium	Function could be marked as external.
61	(SWC-000) Unknown	Medium	Function could be marked as external.
65	(SWC-000) Unknown	Medium	Function could be marked as external.
69	(SWC-000) Unknown	Medium	Function could be marked as external.
73	(SWC-000) Unknown	Medium	Function could be marked as external.
81	(SWC-000) Unknown	Medium	Function could be marked as external.

THANK YOU FOR CHOOSING

// HALBORN