BenQi Finance

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 3rd-23rd, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/24/2021	Gabi Urrutia
0.2	Document Edit	05/26/2021	Ataberk Yavuzer
1.0	Final Edits	05/26/2021	Ataberk Yavuzer
1.1	Remediation Plan	06/17/2021	Ataberk Yavuzer

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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

BenQi Finance engaged Halborn to conduct a security assessment on their Smart contracts beginning on May 3rd, 2021 and ending May 23rd, 2021. The security assessment was scoped to the BenQi Finance and an audit of the security risk and implications regarding the changes introduced by the development team at **BenQi Finance** prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, the most essential aspects were tested and verified to achieve objectives and deliverable 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.

Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned one full time security engineer to audit the security of the smart contracts. The engineers is a 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 as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified few security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

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 review 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 (Hardhat and manual deployments on Ganache.)
- Manual testing with custom Javascript.
- Static Analysis of security for scoped contract, and imported functions (Slither.)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs (MythX.)
- Testnet deployment (Remix IDE.)

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.

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:

- BaseJumpRateModelV2.sol
- CarefulMath.sol
- ComptrollerInterface.sol
- Comptroller.sol
- ComptrollerStorage.sol
- DAIInterestRateModelV3.sol
- EIP20Interface.sol
- EIP20NonStandardInterface.sol
- ErrorReporter.sol
- ExponentialNoError.sol
- Exponential.sol
- InterestRateModel.sol
- JumpRateModel.sol
- JumpRateModelV2.sol
- LegacyInterestRateModel.sol
- LegacyJumpRateModelV2.sol
- Maximillion.sol
- PriceOracle.sol
- QiAvax.sol
- QiErc20Delegate.sol

- QiErc20Delegator.sol
- QiErc20Immutable.sol
- QiErc20.sol
- QiQiLikeDelegate.sol
- QiTokenInterfaces.sol
- QiToken.sol
- Reservoir.sol
- SafeMath.sol
- Timelock.sol
- Unitroller.sol
- WhitePaperInterestRateModel.sol
- Chainlink/AggregatorV2V3Interface.sol
- Chainlink/BenqiChainlinkOracle.sol
- Governance/Benqi.sol
- Governance/GovernorAlpha.sol
- Lens/BenqiLens.sol

Commit ID: b94b5fde3f4ac03a6d73a1308854ae493eaa722a

OUT-OF-SCOPE:
- ComptrollerG1.sol
- ComptrollerG2.sol
- ComptrollerG3.sol
- ComptrollerG4.sol
- ComptrollerG5.sol
- ComptrollerG6.sol
- SimplePriceOracle.sol
- QiDaiDelegate.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	5	2

LIKELIHOOD

IMPACT

(HAL-03) (HAL-04) (HAL-05) (HAL-06)		(HAL-01)		
	(HAL-02)			
(HAL-08)	(HAL-07)			

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING PAYMENT AMOUNT CHECK	Medium	ACCEPTED RISK
MISSING ADDRESS VALIDATION	Low	SOLVED: 06/14/2021
USAGE OF BLOCK.TIMESTAMP	Low	NOT APPLICABLE
OUTDATED PRAGMA VERSION	Low	SOLVED: 06/14/2021
FLOATING PRAGMA	Low	SOLVED: 06/14/2021
USAGE OF INLINE ASSEMBLY	Low	ACCEPTED RISK
WRONG DIVISION ON INTEREST RATE MODEL	Informational	SOLVED: 06/14/2021
LACK OF INPUT VALIDATION	Informational	SOLVED: 06/14/2021
STATIC ANALYSIS	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING PAYMENT AMOUNT CHECK - MEDIUM

Description:

The Benqi Finance protocol includes some financial functions such as **borrow**, **liquidate**, **transfer**, etc. Attackers could use these functions with large amounts in Flash-Loan or similar financial attacks. Other DeFi platforms (The DAO, Spartan, and others) were hacked, and attackers stole a significant amount of funds. We observed during the test that there are no restrictions on large transfers.

Code Location:

Comptroller.sol Lines #330-363

contracts > Comptroller.sol

```
330     function borrowAllowed(address qiToken, address borrower, uint borrowAmount) external returns (uint) {
331         // Pausing is a very serious situation - we revert to sound the alarms
332         require(!borrowGuardianPaused[qiToken], "borrow is paused");
333
334         if (!markets[qiToken].isListed) {
335             return uint(Error.MARKET_NOT_LISTED);
336         }
337
338         if (!markets[qiToken].accountMembership[borrower]) {
339             // only qiTokens may call borrowAllowed if borrower not in market
340             require(msg.sender == qiToken, "sender must be qiToken");
341
342             // attempt to add borrower to the market
343             Error err = addToMarketInternal(QiToken(msg.sender), borrower);
344             if (err != Error.NO_ERROR) {
345                 return uint(err);
346             }
347
348             // it should be impossible to break the important invariant
349             assert(markets[qiToken].accountMembership[borrower]);
350         }
351
352         if (oracle.getUnderlyingPrice(QiToken(qiToken)) == 0) {
353             return uint(Error.PRICE_ERROR);
354         }
355
356
357         uint borrowCap = borrowCaps[qiToken];
358         // Borrow cap of 0 corresponds to unlimited borrowing
359         if (borrowCap != 0) {
360             uint totalBorrows = QiToken(qiToken).totalBorrows();
361             uint nextTotalBorrows = add_(totalBorrows, borrowAmount);
362             require(nextTotalBorrows < borrowCap, "market borrow cap reached");
363         }
```

Figure 1: Missing Payment Amount Check - Comptroller

Comptroller.sol Lines #463-493

contracts > Comptroller.sol

```
463     function liquidateBorrowAllowed(
464         address qiTokenBorrowed,
465         address qiTokenCollateral,
466         address liquidator,
467         address borrower,
468         uint repayAmount) external returns (uint) {
469         // Shh - currently unused
470         liquidator;
471
472         if (!markets[qiTokenBorrowed].isListed || !markets[qiTokenCollateral].isListed) {
473             return uint(Error.MARKET_NOT_LISTED);
474         }
475
476         /* The borrower must have shortfall in order to be liquidatable */
477         (Error err, , uint shortfall) = getAccountLiquidityInternal(borrower);
478         if (err != Error.NO_ERROR) {
479             return uint(err);
480         }
481         if (shortfall == 0) {
482             return uint(Error.INSUFFICIENT_SHORTFALL);
483         }
484
485         /* The liquidator may not repay more than what is allowed by the closeFactor */
486         uint borrowBalance = QiToken(qiTokenBorrowed).borrowBalanceStored(borrower);
487         uint maxClose = mul_ScalarTruncate(Exp({mantissa: closeFactorMantissa}), borrowBalance);
488         if (repayAmount > maxClose) {
489             return uint(Error.TOO_MUCH_REPAY);
490         }
491
492         return uint(Error.NO_ERROR);
493     }
```

Figure 2: Missing Payment Amount Check - Comptroller

Comptroller.sol Lines #595-612

contracts > Comptroller.sol

```
595     function transferAllowed(address qiToken, address src, address dst, uint transferTokens) external returns (uint) {
596         // Pausing is a very serious situation - we revert to sound the alarms
597         require(!transferGuardianPaused, "transfer is paused");
598
599         // Currently the only consideration is whether or not
600         //  the src is allowed to redeem this many tokens
601         uint allowed = redeemAllowedInternal(qiToken, src, transferTokens);
602         if (allowed != uint(Error.NO_ERROR)) {
603             return allowed;
604         }
605
606         // Keep the flywheel moving
607         updateQiSupplyIndex(qiToken);
608         distributeSupplierQi(qiToken, src);
609         distributeSupplierQi(qiToken, dst);
610
611         return uint(Error.NO_ERROR);
612     }
```

Figure 3: Missing Payment Amount Check - Comptroller

Recommendation:

It is recommended to implement a mechanism which controls payment amount.

Remediation Plan:

Accepted Risk: It was not considered appropriate to limit users in transactions.

3.2 (HAL-02) MISSING ADDRESS VALIDATION - LOW

Description:

A few contracts in the Benqi project are missing a safety check inside the constructors. The address-type parameter setter should include a zero-address test; otherwise, the contract's functionality may become inaccessible or tokens burnt forever.

Code Location:

Timelock.sol

contracts > Timelock.sol

```
54    function setPendingAdmin(address pendingAdmin_) public {
55        require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock.");
56        pendingAdmin = pendingAdmin_;
57
58        emit NewPendingAdmin(pendingAdmin);
59    }
60
```

Figure 4: Missing Address Validation - Timelock

Reservoir.sol

contracts > Reservoir.sol

```
32    constructor(uint dripRate_, EIP20Interface token_, address target_) public {
33      dripStart = block.timestamp;
34      dripRate = dripRate_;
35      token = token_;
36      target = target_;
37      dripped = 0;
38    }
39
```

Figure 5: Missing Address Validation - Reservoir

Benqi.sol

```
77    /**
78     * @notice Approve `spender` to transfer up to `amount` from `src`
79     * @dev This will overwrite the approval amount for `spender`
80     *  and is subject to issues noted [here](https://eips.ethereum.org/EIPS/eip-20#approve)
81     * @param spender The address of the account which may transfer tokens
82     * @param rawAmount The number of tokens that are approved (2^256-1 means infinite)
83     * @return Whether or not the approval succeeded
84     */
85    function approve(address spender, uint rawAmount) external returns (bool) {
86        uint96 amount;
87        if (rawAmount == uint(-1)) {
88            amount = uint96(-1);
89        } else {
90            amount = safe96(rawAmount, "Benqi::approve: amount exceeds 96 bits");
91        }
92
93        allowances[msg.sender][spender] = amount;
94
95        emit Approval(msg.sender, spender, amount);
96        return true;
97    }
98
```

Figure 6: Missing Address Validation - Benqi

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

It would be safer to perform additional validation before assigning user-supplied values on contracts. Usage of a whitelist method or a modifier could solve the issue.

For example:

Listing 1

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

Remediation Plan:

Solved: BenQi Finance team implemented necessary controls on the address checks.

3.3 (HAL-03) USAGE OF BLOCK.TIMESTAMP - LOW

Description:

The contracts Timelock.sol, Comptroller.sol, ComptrollerG1.sol, Reservoir.sol and GovernorAlpha.sol use block.timestamp. The global variable block.timestamp does not necessarily hold the current time, and may not be accurate. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. There is no guarantee that the value is correct, only that it is higher than the previous block's timestamp.

Code Location:

Timelock.sol

contracts > Timelock.sol

```
108     function getBlockTimestamp() internal view returns (uint) {
109         // solium-disable-next-line security/no-block-members
110         return block.timestamp;
111     }
112 }
```

Figure 7: Usage of block.timestamp - Timelock

Comptroller.sol

contracts > Comptroller.sol

```
1322     function getBlockTimestamp() public view returns (uint) {
1323         return block.timestamp;
1324     }
```

Figure 8: Usage of block.timestamp - Comptroller

Reservoir.sol

contracts > Reservoir.sol

```
45  function drip() public returns (uint) {
46    // First, read storage into memory
47    EIP20Interface token_ = token;
48    uint reservoirBalance_ = token_.balanceOf(address(this)); // TODO: Verify this is a static call
49    uint dripRate_ = dripRate;
50    uint dripStart_ = dripStart;
51    uint dripped_ = dripped;
52    address target_ = target;
53    uint blockNumber_ = block.timestamp;
```

Figure 9: Usage of block.timestamp - Reservoir

GovernorAlpha.sol

contracts > Governance > GovernorAlpha.sol

```
155         uint startTimestamp = add256(block.timestamp, votingDelay());
156         uint endTimestamp = add256(block.timestamp, add256(votingPeriod(), votingDelay()));
157
```

Figure 10: Usage of block.timestamp - GovernorAlpha

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Use block.number instead of block.timestamp to reduce the risk of MEV attacks. If possible, use an oracle.

Remediation Plan:

Not Applicable: The timestamp keyword has been used intentionally. It is not necessary to change the contract given the risk. The consensus protocols of the Avalanche chain will prevent this attack.

3.4 (HAL-04) PRAGMA VERSION - LOW

Description:

In the Benqi contracts use pragma version 0.5.16. The latest pragma version is (0.8.3) was released in April 2021. Many pragma versions have been released, going from version 0.6.x to the recently released version 0.8.x. in just 6 months.

Code Location:

QiToken.sol Line #1

contracts > QiToken.sol

```
1  pragma solidity ^0.5.16;
2
3  import "./ComptrollerInterface.sol";
4  import "./QiTokenInterfaces.sol";
5  import "./ErrorReporter.sol";
6  import "./Exponential.sol";
7  import "./EIP20Interface.sol";
8  import "./InterestRateModel.sol";
```

Figure 11: Pragma Version - QiToken

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

In the Solidity Github repository, there is a json file listing the bugs reported for each compiler version. No bugs have been found in > 0.7.3 versions and very few in 0.7.0 -- 0.7.3. The latest stable version is pragma 0.6.12. Furthermore, pragma 0.6.12 is widely used by Solidity developers and has been extensively tested in many security audits. We recommend using at minimum the latest stable version.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Remediation Plan:

Solved: The BenQi Finance team fixed this issue by setting the pragma version to 0.5.17 which is one of the stable releases.

3.5 (HAL-05) FLOATING PRAGMA - LOW

Benqi contracts use the floating pragma ^0.5.12. 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:

QiToken.sol Line #1

contracts > QiToken.sol

```
1  pragma solidity ^0.5.16;
2
3  import "./ComptrollerInterface.sol";
4  import "./QiTokenInterfaces.sol";
5  import "./ErrorReporter.sol";
6  import "./Exponential.sol";
7  import "./EIP20Interface.sol";
8  import "./InterestRateModel.sol";
```

- This is an example when we can find the use of floating pragma ^0.5.12.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Lock the pragma version whenever possible and avoid using a floating pragma in the final deployment. The pragma can be locked in the code by removing the caret (^) and by specifying the exact version in the Truffle configuration file truffle-config.js or hardhat.config.js if using the HardHat framework.

Remediation Plan:

Solved: The BenQi Finance team solved the issue by locking the pragma on the contracts.

3.6 (HAL-06) USAGE OF INLINE ASSEMBLY - LOW

Description:

Inline assembly is a way to access the Virtual Machine at a low level. This discards several important safety features in Solidity.

Benqi.sol Line #297

contracts > Governance > Benqi.sol

```
295     function getChainId() internal pure returns (uint) {
296         uint256 chainId;
297         assembly { chainId := chainid() }
298         return chainId;
299     }
```

QiErc20.sol Lines #158-170

contracts > QiErc20.sol

```
158         assembly {
159             switch returndatasize()
160                 case 0 {                      // This is a non-standard ERC-20
161                     success := not(0)         // set success to true
162                 }
163                 case 32 {                     // This is a compliant ERC-20
164                     returndatacopy(0, 0, 32)
165                     success := mload(0)       // Set `success = returndata` of external call
166                 }
167                 default {                     // This is an excessively non-compliant ERC-20, revert.
168                     revert(0, 0)
169                 }
170         }
```

QiErc20Delegator.sol Lines #449-453

contracts > QiErc20Delegator.sol

```
422         assembly {
423             if eq(success, 0) {
424                 revert(add(returnData, 0x20), returndatasize)
425             }
426         }
```

Unitroller.sol Lines #139-146

contracts > Unitroller.sol

```
139         assembly {
140             let free_mem_ptr := mload(0x40)
141             returndatacopy(free_mem_ptr, 0, returndatasize)
142
143             switch success
144             case 0 { revert(free_mem_ptr, returndatasize) }
145             default { return(free_mem_ptr, returndatasize) }
146         }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

The contracts should avoid using inline assembly because it interacts with the AVM (Avalanche Virtual Machine) at a low level. An attacker could bypass many essential safety features of Solidity.

Remediation Plan:

Accepted Risk: The contracts require inline assembly, and the risk is acceptable.

3.7 (HAL-07) INCORRECT DIVISION ON WHITEPAPER INTEREST RATE MODEL - INFORMATIONAL

Description:

In the WhitePaperInterestRateModel.sol contract, there is an incorrect division operation. According to comments in the contract, the interest rate is raised by 1e18. Some functions multiply user-controlled values by 1e18. However, the constructor doesn't have this multiplication operation. The **timestampsPerYear** variable is defined as 31536000.When the contract is deployed, to set "200" as the baseRatePerTimestamp variable, the deployer must use 200 * 31536000 for the baseRatePerYear variable. If the deployer sets the baseRatePerYear variable to "200", then the contract will execute the following code to calculate baseRatePerTimestamp variable:

Listing 2

```
1. baseRatePerTimestamp = baseRatePerYear.div(timestampsPerYear);
2. baseRatePerTimestamp = 200.div(31536000);
3. baseRatePerTimestamp = 0
```

Code Location:

WhitePaperInterestRateModel.sol Line #19-41

contracts > WhitePaperInterestRateModel.sol

```
19      uint public constant timestampsPerYear = 31536000;
20
21      /**
22       * @notice The multiplier of utilization rate that gives the slope of the interest rate
23       */
24      uint public multiplierPerTimestamp;
25
26      /**
27       * @notice The base interest rate which is the y-intercept when utilization rate is 0
28       */
29      uint public baseRatePerTimestamp;
30
31      /**
32       * @notice Construct an interest rate model
33       * @param baseRatePerYear The approximate target base APR, as a mantissa (scaled by 1e18)
34       * @param multiplierPerYear The rate of increase in interest rate wrt utilization (scaled by 1e18)
35       */
36      constructor(uint baseRatePerYear, uint multiplierPerYear) public {
37          baseRatePerTimestamp = baseRatePerYear.div(timestampsPerYear);
38          multiplierPerTimestamp = multiplierPerYear.div(timestampsPerYear);
39
40          emit NewInterestParams(baseRatePerTimestamp, multiplierPerTimestamp);
41      }
42
```

Figure 12: Incorrect Division on Interest Rate Model

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Multiply the baseRatePerYear and multiplierPerYear variables as a mantissa value raised by 1e18 power before division. Otherwise, the result will be zero.

Remediation Plan:

Solved: The BenQi Finance team solved this issue by adding new multiplication and division operations to the related functions.

3.8 (HAL-08) LACK OF INPUT VALIDATION - INFORMATIONAL

Description:

In the GovernorAlpha.sol contract, The description parameter can be left empty by the function.

Code Location:

GovernanceAlpha.sol Line #142

```
142    function propose(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas, string memory description) public returns (
143        require(benqi.getPriorVotes(msg.sender, sub256(block.number, 1)) > proposalThreshold(), "GovernorAlpha::propose: proposer votes below proposal threshold");
144        require(targets.length == values.length && targets.length == signatures.length && targets.length == calldatas.length, "GovernorAlpha::propose: proposal functio
145        require(targets.length != 0, "GovernorAlpha::propose: must provide actions");
146        require(targets.length <= proposalMaxOperations(), "GovernorAlpha::propose: too many actions");
147
148        uint latestProposalId = latestProposalIds[msg.sender];
149        if (latestProposalId != 0) {
150          ProposalState proposersLatestProposalState = state(latestProposalId);
151          require(proposersLatestProposalState != ProposalState.Active, "GovernorAlpha::propose: one live proposal per proposer, found an already active proposal");
152          require(proposersLatestProposalState != ProposalState.Pending, "GovernorAlpha::propose: one live proposal per proposer, found an already pending proposal");
153        }
154
155        uint startTimestamp = add256(block.timestamp, votingDelay());
156        uint endTimestamp = add256(block.timestamp, add256(votingPeriod(), votingDelay()));
157
158        proposalCount++;
159        Proposal memory newProposal = Proposal({
160            id: proposalCount,
161            proposer: msg.sender,
162            eta: 0,
163            targets: targets,
164            values: values,
165            signatures: signatures,
166            calldatas: calldatas,
167            startTimestamp: startTimestamp,
168            endTimestamp: endTimestamp,
169            startBlock: 0,
170            forVotes: 0,
171            againstVotes: 0,
172            canceled: false,
173            executed: false
174        });
175
176        proposals[newProposal.id] = newProposal;
177        latestProposalIds[newProposal.proposer] = newProposal.id;
```

Figure 13: Lack Of Input Validation

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The contract should implement validation on the user controlled inputs which should be performed using a require statement.

Remediation Plan:

Solved: The BenQi Finance team fixed this issue by implementing input validation to the contract.

3.9 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:

Comptroller.sol

```
INFO:Detectors:
Comptroller (contracts/Comptroller.sol#15-1333) contract sets array length with a user-controlled value:
        - allMarkets.push(QiToken(qiToken)) (contracts/Comptroller.sol#941)
Comptroller (contracts/Comptroller.sol#15-1333) contract sets array length with a user-controlled value:
        - accountAssets[borrower].push(qiToken) (contracts/Comptroller.sol#149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#array-length-assignment
INFO:Detectors:
Unitroller.fallback() (contracts/Unitroller.sol#135-147) uses delegatecall to a input-controlled function id
        - (success) = comptrollerImplementation.delegatecall(msg.data) (contracts/Unitroller.sol#137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
Comptroller.grantQiInternal(address,uint256) (contracts/Comptroller.sol#1257-1265) ignores return value by benqi.transfer(user,amount) (contracts/Comptroller.sol#1261)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
UnitrollerAdminStorage.comptrollerImplementation (contracts/ComptrollerStorage.sol#20) is never initialized. It is used in:
        - Comptroller.adminOrInitializing() (contracts/Comptroller.sol#1050-1052)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
Comptroller.mintVerify(address,address,uint256,uint256) (contracts/Comptroller.sol#250-261) uses a Boolean constant improperly:
        -false (contracts/Comptroller.sol#258)
Comptroller.borrowVerify(address,address,uint256) (contracts/Comptroller.sol#387-397) uses a Boolean constant improperly:
        -false (contracts/Comptroller.sol#394)
Comptroller.repayBorrowVerify(address,address,address,uint256,uint256) (contracts/Comptroller.sol#436-453) uses a Boolean constant improperly:
        -false (contracts/Comptroller.sol#450)
Comptroller.liquidateBorrowVerify(address,address,address,address,uint256,uint256) (contracts/Comptroller.sol#503-522) uses a Boolean constant improperly:
        -false (contracts/Comptroller.sol#519)
Comptroller.seizeVerify(address,address,address,address,uint256) (contracts/Comptroller.sol#568-585) uses a Boolean constant improperly:
        -false (contracts/Comptroller.sol#582)
Comptroller.transferVerify(address,address,address,uint256) (contracts/Comptroller.sol#621-632) uses a Boolean constant improperly:
        -false (contracts/Comptroller.sol#629)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#misuse-of-a-boolean-constant
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
Comptroller._grantQi(address,uint256) (contracts/Comptroller.sol#1275-1280) uses a dangerous strict equality:
        - require(bool,string)(amountLeft == 0,Insufficient benqi for grant) (contracts/Comptroller.sol#1278)
Comptroller.getHypotheticalAccountLiquidityInternal(address,QiToken,uint256,uint256) (contracts/Comptroller.sol#707-763) uses a dangerous strict equality:
        - vars.oraclePriceMantissa == 0 (contracts/Comptroller.sol#731)
Comptroller.liquidateBorrowAllowed(address,address,address,address,uint256) (contracts/Comptroller.sol#463-493) uses a dangerous strict equality:
        - shortfall == 0 (contracts/Comptroller.sol#481)
ExponentialNoError.mul_(uint256,uint256,string) (contracts/ExponentialNoError.sol#150-157) uses a dangerous strict equality:
        - a == 0 || b == 0 (contracts/ExponentialNoError.sol#151)
ExponentialNoError.mul_(uint256,uint256,string) (contracts/ExponentialNoError.sol#150-157) uses a dangerous strict equality:
        - require(bool,string)(c / a == b,errorMessage) (contracts/ExponentialNoError.sol#155)
Comptroller.setQiSpeedInternal(QiToken,uint256) (contracts/Comptroller.sol#1061-1092) uses a dangerous strict equality:
        - qiSupplyState[address(qiToken)].index == 0 && qiSupplyState[address(qiToken)].timestamp == 0 (contracts/Comptroller.sol#1073)
Benqi._writeCheckpoint(address,uint32,uint96,uint96) (contracts/Governance/Benqi.sol#261-272) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (contracts/Governance/Benqi.sol#264)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
```

CarefulMath.sol

```
INFO:Detectors:
CarefulMath.addThenSubUInt(uint256,uint256,uint256) (contracts/CarefulMath.sol#76-84) is never used and should be removed
CarefulMath.addUInt(uint256,uint256) (contracts/CarefulMath.sol#63-71) is never used and should be removed
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) is never used and should be removed
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) is never used and should be removed
CarefulMath.subUInt(uint256,uint256) (contracts/CarefulMath.sol#52-58) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Slither:contracts/CarefulMath.sol analyzed (1 contracts with 75 detectors), 5 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Chainlink/BenqiChainlinkOracle.sol

```
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardInterface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/QiToken.sol#406)
QiToken.balanceOfUnderlying(address) (contracts/QiToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/QiToken.sol#193)
QiToken.borrowBalanceStored(address) (contracts/QiToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/QiToken.sol#273)
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
QiToken.exchangeRateStored() (contracts/QiToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/QiToken.sol#330)
QiToken.exchangeRateStoredInternal() (contracts/QiToken.sol#339-369) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/QiToken.sol#341)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (contracts/QiToken.sol#32)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/QiToken.sol#40)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/QiToken.sol#48)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/QiToken.sol#1000)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/QiToken.sol#1014)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/QiToken.sol#536)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/QiToken.sol#544)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#547)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#901)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#904)
QiToken.transfer(address,uint256) (contracts/QiToken.sol#135-137) uses a dangerous strict equality:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#136)
QiToken.transferFrom(address,address,uint256) (contracts/QiToken.sol#146-148) uses a dangerous strict equality:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in QiToken.liquidateBorrowInternal(address,uint256,QiTokenInterface) (contracts/QiToken.sol#929-944):
        External calls:
        - error = qiTokenCollateral.accrueInterest() (contracts/QiToken.sol#936)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,qiTokenCollateral) (contracts/QiToken.sol#943)
```

BaseJumpRateModelV2.sol

```
INFO:Detectors:
BaseJumpRateModelV2.getSupplyRate(uint256,uint256,uint256,uint256) (contracts/BaseJumpRateModelV2.sol#115-120) performs a multiplication on the result of a division:
        -rateToPool = borrowRate.mul(oneMinusReserveFactor).div(1e18) (contracts/BaseJumpRateModelV2.sol#118)
        -utilizationRate(cash,borrows,reserves).mul(rateToPool).div(1e18) (contracts/BaseJumpRateModelV2.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
SafeMath.add(uint256,uint256,string) (contracts/SafeMath.sol#43-48) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/SafeMath.sol#167-169) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/SafeMath.sol#182-185) is never used and should be removed
SafeMath.mul(uint256,uint256,string) (contracts/SafeMath.sol#107-119) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Constant BaseJumpRateModelV2.timestampsPerYear (contracts/BaseJumpRateModelV2.sol#23) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
getSupplyRate(uint256,uint256,uint256,uint256) should be declared external:
        - BaseJumpRateModelV2.getSupplyRate(uint256,uint256,uint256,uint256) (contracts/BaseJumpRateModelV2.sol#115-120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/BaseJumpRateModelV2.sol analyzed (2 contracts with 75 detectors), 7 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Governance/Benqi.sol

```
INFO:Detectors:
Benqi._writeCheckpoint(address,uint32,uint96,uint96) (contracts/Governance/Benqi.sol#261-272) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (contracts/Governance/Benqi.sol#264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Benqi.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (contracts/Governance/Benqi.sol#160-169) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,Benqi::delegateBySig: signature expired) (contracts/Governance/Benqi.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Benqi.getChainId() (contracts/Governance/Benqi.sol#295-299) uses assembly
        - INLINE ASM (contracts/Governance/Benqi.sol#297)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Constant Benqi.totalSupply (contracts/Governance/Benqi.sol#14) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Benqi.slitherConstructorConstantVariables() (contracts/Governance/Benqi.sol#3-300) uses literals with too many digits:
        - totalSupply = 720000000e18 (contracts/Governance/Benqi.sol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
delegate(address) should be declared external:
        - Benqi.delegate(address) (contracts/Governance/Benqi.sol#147-149)
delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - Benqi.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (contracts/Governance/Benqi.sol#160-169)
getPriorVotes(address,uint256) should be declared external:
        - Benqi.getPriorVotes(address,uint256) (contracts/Governance/Benqi.sol#188-220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/Governance/Benqi.sol analyzed (1 contracts with 75 detectors), 8 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Governance/GovernorAlpha.sol

```
INFO:Detectors:
GovernorAlpha.execute(uint256) (contracts/Governance/GovernorAlpha.sol#199-207) sends eth to arbitrary user
        Dangerous calls:
        - timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i],proposal.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/Governance/GovernorAlpha.sol#204
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
GovernorAlpha._queueOrRevert(address,uint256,string,bytes,uint256) (contracts/Governance/GovernorAlpha.sol#194-197) ignores return value by timelock.queueTransaction(target,value,signature,data,eta) (contr
acts/Governance/GovernorAlpha.sol#196)
GovernorAlpha.execute(uint256) (contracts/Governance/GovernorAlpha.sol#199-207) ignores return value by timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i],proposal
.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/Governance/GovernorAlpha.sol#204)
GovernorAlpha.__queueSetTimelockPendingAdmin(address,uint256) (contracts/Governance/GovernorAlpha.sol#302-305) ignores return value by timelock.queueTransaction(address(timelock),0,setPendingAdmin(address)
,abi.encode(newPendingAdmin),eta) (contracts/Governance/GovernorAlpha.sol#304)
GovernorAlpha.__executeSetTimelockPendingAdmin(address,uint256) (contracts/Governance/GovernorAlpha.sol#307-310) ignores return value by timelock.executeTransaction(address(timelock),0,setPendingAdmin(addr
ess),abi.encode(newPendingAdmin),eta) (contracts/Governance/GovernorAlpha.sol#309)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
GovernorAlpha.cancel(uint256).state (contracts/Governance/GovernorAlpha.sol#210) shadows:
        - GovernorAlpha.state(uint256) (contracts/Governance/GovernorAlpha.sol#233-253) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
GovernorAlpha.constructor(address,address,address).guardian_ (contracts/Governance/GovernorAlpha.sol#136) lacks a zero-check on :
                - guardian = guardian_ (contracts/Governance/GovernorAlpha.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
GovernorAlpha.execute(uint256) (contracts/Governance/GovernorAlpha.sol#199-207) has external calls inside a loop: timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i
],proposal.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/Governance/GovernorAlpha.sol#204)
GovernorAlpha.cancel(uint256) (contracts/Governance/GovernorAlpha.sol#209-222) has external calls inside a loop: timelock.cancelTransaction(proposal.targets[i],proposal.values[i],proposal.signatures[i],pro
posal.calldatas[i],proposal.eta) (contracts/Governance/GovernorAlpha.sol#218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
GovernorAlpha._queueOrRevert(address,uint256,string,bytes,uint256) (contracts/Governance/GovernorAlpha.sol#194-197) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(! timelock.queuedTransactions(keccak256(bytes)(abi.encode(target,value,signature,data,eta))),GovernorAlpha::_queueOrRevert: proposal action already queued at eta) (contracts/
Governance/GovernorAlpha.sol#195)
GovernorAlpha.state(uint256) (contracts/Governance/GovernorAlpha.sol#233-253) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp <= proposal.startTimestamp (contracts/Governance/GovernorAlpha.sol#238)
        - block.timestamp <= proposal.endTimestamp (contracts/Governance/GovernorAlpha.sol#240)
        - block.timestamp >= add256(proposal.eta,timelock.GRACE_PERIOD()) (contracts/Governance/GovernorAlpha.sol#248)
GovernorAlpha.add256(uint256,uint256) (contracts/Governance/GovernorAlpha.sol#312-316) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(c >= a,addition overflow) (contracts/Governance/GovernorAlpha.sol#314)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
GovernorAlpha.getChainId() (contracts/Governance/GovernorAlpha.sol#323-327) uses assembly
        - INLINE ASM (contracts/Governance/GovernorAlpha.sol#325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
GovernorAlpha._castVote(address,uint256,bool) (contracts/Governance/GovernorAlpha.sol#268-290) compares to a boolean constant:
        -require(bool,string)(receipt.hasVoted == false,GovernorAlpha::_castVote: voter already voted) (contracts/Governance/GovernorAlpha.sol#276)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Function GovernorAlpha.__acceptAdmin() (contracts/Governance/GovernorAlpha.sol#292-295) is not in mixedCase
Function GovernorAlpha.__abdicate() (contracts/Governance/GovernorAlpha.sol#297-300) is not in mixedCase
Function GovernorAlpha.__queueSetTimelockPendingAdmin(address,uint256) (contracts/Governance/GovernorAlpha.sol#302-305) is not in mixedCase
Function GovernorAlpha.__executeSetTimelockPendingAdmin(address,uint256) (contracts/Governance/GovernorAlpha.sol#307-310) is not in mixedCase
Function TimelockInterface.GRACE_PERIOD() (contracts/Governance/GovernorAlpha.sol#332) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
GovernorAlpha.quorumVotes() (contracts/Governance/GovernorAlpha.sol#9) uses literals with too many digits:
```

Maxmillion.sol

```
INFO:Detectors:
Maximillion.repayBehalfExplicit(address,QiAvax) (contracts/Maximillion.sol#37-46) sends eth to arbitrary user
        Dangerous calls:
        - qiAvax_.repayBorrowBehalf.value(borrows)(borrower) (contracts/Maximillion.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
QiAvax.requireNoError(uint256,string) (contracts/QiAvax.sol#148-167) uses a weak PRNG: "fullMessage[i + 3] = bytes1(uint8(48 + (errCode % 10)))" (contracts/QiAvax.sol#163)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/QiToken.sol#406)
QiToken.balanceOfUnderlying(address) (contracts/QiToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/QiToken.sol#193)
QiToken.borrowBalanceStored(address) (contracts/QiToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/QiToken.sol#273)
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
QiAvax.doTransferIn(address,uint256) (contracts/QiAvax.sol#136-141) uses a dangerous strict equality:
        - require(bool,string)(msg.value == amount,value mismatch) (contracts/QiAvax.sol#139)
QiToken.exchangeRateStored() (contracts/QiToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/QiToken.sol#330)
QiToken.exchangeRateStoredInternal() (contracts/QiToken.sol#339-369) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/QiToken.sol#341)
QiAvax.getCashPrior() (contracts/QiAvax.sol#124-128) uses a dangerous strict equality:
        - require(bool)(err == MathError.NO_ERROR) (contracts/QiAvax.sol#126)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (contracts/QiToken.sol#32)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/QiToken.sol#40)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/QiToken.sol#48)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/QiToken.sol#1000)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/QiToken.sol#1014)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/QiToken.sol#536)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/QiToken.sol#544)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#547)
Exponential.mulExp(Exponential.Exp,Exponential.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#901)
```

PriceOracle.sol

```
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/QiToken.sol#406)
QiToken.balanceOfUnderlying(address) (contracts/QiToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/QiToken.sol#193)
QiToken.borrowBalanceStored(address) (contracts/QiToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/QiToken.sol#273)
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
QiToken.exchangeRateStored() (contracts/QiToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/QiToken.sol#330)
QiToken.exchangeRateStoredInternal() (contracts/QiToken.sol#339-369) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/QiToken.sol#341)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (contracts/QiToken.sol#32)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/QiToken.sol#40)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/QiToken.sol#48)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/QiToken.sol#1000)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/QiToken.sol#1014)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/QiToken.sol#536)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/QiToken.sol#544)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#547)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#901)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#904)
QiToken.transfer(address,uint256) (contracts/QiToken.sol#135-137) uses a dangerous strict equality:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#136)
QiToken.transferFrom(address,address,uint256) (contracts/QiToken.sol#146-148) uses a dangerous strict equality:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in QiToken.liquidateBorrowInternal(address,uint256,QiTokenInterface) (contracts/QiToken.sol#929-944):
        External calls:
        - error = qiTokenCollateral.accrueInterest() (contracts/QiToken.sol#936)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,qiTokenCollateral) (contracts/QiToken.sol#943)
```

QiAvax.sol

```
INFO:Detectors:
QiAvax.requireNoError(uint256,string) (contracts/QiAvax.sol#148-167) uses a weak PRNG: "fullMessage[i + 3] = bytes1(uint8(48 + (errCode % 10))) (contracts/QiAvax.sol#163)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/QiToken.sol#406)
QiToken.balanceOfUnderlying(address) (contracts/QiToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/QiToken.sol#193)
QiToken.borrowBalanceStored(address) (contracts/QiToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/QiToken.sol#273)
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
QiAvax.doTransferIn(address,uint256) (contracts/QiAvax.sol#136-141) uses a dangerous strict equality:
        - require(bool,string)(msg.value == amount,value mismatch) (contracts/QiAvax.sol#139)
QiToken.exchangeRateStored() (contracts/QiToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/QiToken.sol#330)
QiToken.exchangeRateStoredInternal() (contracts/QiToken.sol#339-369) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/QiToken.sol#341)
QiAvax.getCashPrior() (contracts/QiAvax.sol#124-128) uses a dangerous strict equality:
        - require(bool)(err == MathError.NO_ERROR) (contracts/QiAvax.sol#126)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (contracts/QiToken.sol#32)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/QiToken.sol#40)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/QiToken.sol#48)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/QiToken.sol#1000)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/QiToken.sol#1014)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/QiToken.sol#536)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/QiToken.sol#544)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#547)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#901)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#904)
QiAvax.requireNoError(uint256,string) (contracts/QiAvax.sol#148-167) uses a dangerous strict equality:
        - errCode == uint256(Error.NO_ERROR) (contracts/QiAvax.sol#149)
QiAvax.requireNoError(uint256,string) (contracts/QiAvax.sol#148-167) uses a dangerous strict equality:
```

QiToken.sol

```
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/QiToken.sol#406)
QiToken.balanceOfUnderlying(address) (contracts/QiToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/QiToken.sol#193)
QiToken.borrowBalanceStored(address) (contracts/QiToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/QiToken.sol#273)
CarefulMath.divUint(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
QiToken.exchangeRateStored() (contracts/QiToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/QiToken.sol#330)
QiToken.exchangeRateStoredInternal() (contracts/QiToken.sol#339-369) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/QiToken.sol#341)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (contracts/QiToken.sol#32)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/QiToken.sol#40)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/QiToken.sol#48)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/QiToken.sol#1000)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/QiToken.sol#1014)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/QiToken.sol#536)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/QiToken.sol#544)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#547)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUint(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#901)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#904)
QiToken.transfer(address,uint256) (contracts/QiToken.sol#135-137) uses a dangerous strict equality:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#136)
QiToken.transferFrom(address,address,uint256) (contracts/QiToken.sol#146-148) uses a dangerous strict equality:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in QiToken.liquidateBorrowInternal(address,uint256,QiTokenInterface) (contracts/QiToken.sol#929-944):
        External calls:
        - error = qiTokenCollateral.accrueInterest() (contracts/QiToken.sol#936)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,qiTokenCollateral) (contracts/QiToken.sol#943)
```

QiErc20Delegator.sol

```
INFO:Detectors:
QiErc20Delegator.delegateTo(address,bytes) (contracts/QiErc20Delegator.sol#420-428) uses delegatecall to a input-controlled function id
        - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
QiErc20Delegator.fallback() (contracts/QiErc20Delegator.sol#461-475) uses delegatecall to a input-controlled function id
        - (success) = implementation.delegatecall(msg.data) (contracts/QiErc20Delegator.sol#465)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
Reentrancy in QiErc20Delegator._setImplementation(address,bool,bytes) (contracts/QiErc20Delegator.sol#60-73):
        External calls:
        - delegateToImplementation(abi.encodeWithSignature(_resignImplementation())) (contracts/QiErc20Delegator.sol#64)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        State variables written after the call(s):
        - implementation = implementation_ (contracts/QiErc20Delegator.sol#68)
Reentrancy in QiErc20Delegator.constructor(address,ComptrollerInterface,InterestRateModel,uint256,string,string,uint8,address,address,bytes) (contracts/QiErc20Delegator.sol#24-52):
        External calls:
        - delegateTo(implementation_,abi.encodeWithSignature(initialize(address,address,address,uint256,string,string,uint8),underlying_,comptroller_,interestRateModel_,initialExchangeRateMantissa_,name_,s
ymbol_,decimals_)) (contracts/QiErc20Delegator.sol#38-45)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        - _setImplementation(implementation_,false,becomeImplementationData) (contracts/QiErc20Delegator.sol#48)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        State variables written after the call(s):
        - admin = admin_ (contracts/QiErc20Delegator.sol#51)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
QiErc20Delegator.constructor(address,ComptrollerInterface,InterestRateModel,uint256,string,string,uint8,address,address,bytes).admin_ (contracts/QiErc20Delegator.sol#31) lacks a zero-check on :
                - admin = admin_ (contracts/QiErc20Delegator.sol#51)
QiErc20Delegator._setImplementation(address,bool,bytes).implementation_ (contracts/QiErc20Delegator.sol#60) lacks a zero-check on :
                - implementation = implementation_ (contracts/QiErc20Delegator.sol#68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in QiErc20Delegator._setImplementation(address,bool,bytes) (contracts/QiErc20Delegator.sol#60-73):
        External calls:
        - delegateToImplementation(abi.encodeWithSignature(_resignImplementation())) (contracts/QiErc20Delegator.sol#64)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        - delegateToImplementation(abi.encodeWithSignature(_becomeImplementation(bytes),becomeImplementationData)) (contracts/QiErc20Delegator.sol#70)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        Event emitted after the call(s):
        - NewImplementation(oldImplementation,implementation) (contracts/QiErc20Delegator.sol#72)
Reentrancy in QiErc20Delegator.constructor(address,ComptrollerInterface,InterestRateModel,uint256,string,string,uint8,address,address,bytes) (contracts/QiErc20Delegator.sol#24-52):
        External calls:
        - delegateTo(implementation_,abi.encodeWithSignature(initialize(address,address,address,uint256,string,string,uint8),underlying_,comptroller_,interestRateModel_,initialExchangeRateMantissa_,name_,s
ymbol_,decimals_)) (contracts/QiErc20Delegator.sol#38-45)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        - _setImplementation(implementation_,false,becomeImplementationData) (contracts/QiErc20Delegator.sol#48)
                - (success,returnData) = callee.delegatecall(data) (contracts/QiErc20Delegator.sol#421)
        Event emitted after the call(s):
        - NewImplementation(oldImplementation,implementation) (contracts/QiErc20Delegator.sol#72)
                - _setImplementation(implementation_,false,becomeImplementationData) (contracts/QiErc20Delegator.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
QiErc20Delegator.delegateTo(address,bytes) (contracts/QiErc20Delegator.sol#420-428) uses assembly
```

Timelock.sol

```
INFO:Detectors:
Timelock.constructor(address,uint256).admin_ (contracts/Timelock.sol#26) lacks a zero-check on :
                - admin = admin_ (contracts/Timelock.sol#30)
Timelock.setPendingAdmin(address).pendingAdmin_ (contracts/Timelock.sol#53) lacks a zero-check on :
                - pendingAdmin = pendingAdmin_ (contracts/Timelock.sol#55)
Timelock.executeTransaction(address,uint256,string,bytes,uint256).target (contracts/Timelock.sol#80) lacks a zero-check on :
                - (success,returnData) = target.call.value(value)(callData) (contracts/Timelock.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#80-105):
        External calls:
        - (success,returnData) = target.call.value(value)(callData) (contracts/Timelock.sol#99)
        Event emitted after the call(s):
        - ExecuteTransaction(txHash,target,value,signature,data,eta) (contracts/Timelock.sol#102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Timelock.queueTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#60-69) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(eta >= getBlockTimestamp().add(delay),Timelock::queueTransaction: Estimated execution block must satisfy delay.) (contracts/Timelock.sol#62)
Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#80-105) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(getBlockTimestamp() >= eta,Timelock::executeTransaction: Transaction hasn't surpassed time lock.) (contracts/Timelock.sol#85)
        - require(bool,string)(getBlockTimestamp() <= eta.add(GRACE_PERIOD),Timelock::executeTransaction: Transaction is stale.) (contracts/Timelock.sol#86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
SafeMath.add(uint256,uint256,string) (contracts/SafeMath.sol#43-48) is never used and should be removed
SafeMath.div(uint256,uint256) (contracts/SafeMath.sol#132-134) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/SafeMath.sol#147-154) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/SafeMath.sol#167-169) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/SafeMath.sol#182-185) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/SafeMath.sol#85-97) is never used and should be removed
SafeMath.mul(uint256,uint256,string) (contracts/SafeMath.sol#107-119) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/SafeMath.sol#58-60) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (contracts/SafeMath.sol#70-75) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#80-105):
        - (success,returnData) = target.call.value(value)(callData) (contracts/Timelock.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
setDelay(uint256) should be declared external:
        - Timelock.setDelay(uint256) (contracts/Timelock.sol#36-43)
acceptAdmin() should be declared external:
        - Timelock.acceptAdmin() (contracts/Timelock.sol#45-51)
setPendingAdmin(address) should be declared external:
        - Timelock.setPendingAdmin(address) (contracts/Timelock.sol#53-58)
queueTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.queueTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#60-69)
cancelTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#71-78)
executeTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#80-105)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Unitroller.sol

```
INFO:Detectors:
Unitroller.fallback() (contracts/Unitroller.sol#135-147) uses delegatecall to a input-controlled function id
        - (success) = comptrollerImplementation.delegatecall(msg.data) (contracts/Unitroller.sol#137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#
34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardIn
terface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (contracts/QiToken.sol#390)
QiToken.accrueInterest() (contracts/QiToken.sol#384-462) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/QiToken.sol#406)
QiToken.balanceOfUnderlying(address) (contracts/QiToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/QiToken.sol#193)
QiToken.borrowBalanceStored(address) (contracts/QiToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/QiToken.sol#273)
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
QiToken.exchangeRateStored() (contracts/QiToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/QiToken.sol#330)
QiToken.exchangeRateStoredInternal() (contracts/QiToken.sol#339-369) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/QiToken.sol#341)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (contracts/QiToken.sol#32)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/QiToken.sol#40)
QiToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/QiToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/QiToken.sol#48)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/QiToken.sol#1008)
QiToken.liquidateBorrowFresh(address,address,uint256,QiTokenInterface) (contracts/QiToken.sol#955-1024) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/QiToken.sol#1014)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/QiToken.sol#536)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/QiToken.sol#544)
QiToken.mintFresh(address,uint256) (contracts/QiToken.sol#497-562) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#547)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#901)
QiToken.repayBorrowFresh(address,address,uint256) (contracts/QiToken.sol#852-919) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (contracts/QiToken.sol#904)
QiToken.transfer(address,uint256) (contracts/QiToken.sol#135-137) uses a dangerous strict equality:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#136)
QiToken.transferFrom(address,address,uint256) (contracts/QiToken.sol#146-148) uses a dangerous strict equality:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (contracts/QiToken.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
```

WhitePaperInterestModel.sol

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

3.10 AUTOMATED SECURITY SCAN

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. Only security-related findings were considered as in-scope.

Results:

Comptroller.sol

Report for Comptroller.sol
https://dashboard.mythx.io/#/console/analyses/ba61fc34-dde6-4f9a-9acf-0c665ffc8c63

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
15	(SWC-123) Requirement Violation	Low	Requirement violation.
91	(SWC-128) DoS With Block Gas Limit	Low	Implicit loop over unbounded data structure.
111	(SWC-000) Unknown	Medium	Function could be marked as external.
192	(SWC-128) DoS With Block Gas Limit	Medium	Implicit loop over unbounded data structure.
195	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
660	(SWC-000) Unknown	Medium	Function could be marked as external.
686	(SWC-000) Unknown	Medium	Function could be marked as external.
717	(SWC-128) DoS With Block Gas Limit	Medium	Implicit loop over unbounded data structure.
718	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
809	(SWC-000) Unknown	Medium	Function could be marked as external.
925	(SWC-123) Requirement Violation	Low	Requirement violation.
938	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
987	(SWC-000) Unknown	Medium	Function could be marked as external.
1004	(SWC-000) Unknown	Medium	Function could be marked as external.
1014	(SWC-000) Unknown	Medium	Function could be marked as external.
1024	(SWC-000) Unknown	Medium	Function could be marked as external.
1033	(SWC-000) Unknown	Medium	Function could be marked as external.
1042	(SWC-000) Unknown	Medium	Function could be marked as external.
1105	(SWC-119) Shadowing State Variables	Low	Local variable shadows a state variable.
1128	(SWC-119) Shadowing State Variables	Low	Local variable shadows a state variable.
1206	(SWC-000) Unknown	Medium	Function could be marked as external.
1207	(SWC-128) DoS With Block Gas Limit	Medium	Implicit loop over unbounded data structure.
1275	(SWC-000) Unknown	Medium	Function could be marked as external.
1287	(SWC-000) Unknown	Medium	Function could be marked as external.
1297	(SWC-000) Unknown	Medium	Function could be marked as external.
1318	(SWC-000) Unknown	Medium	Function could be marked as external.
1319	(SWC-128) DoS With Block Gas Limit	Low	Implicit loop over unbounded data structure.

CarefulMath.sol

Report for contracts/CarefulMath.sol
https://dashboard.mythx.io/#/console/analyses/10d2e68f-8ae8-42c2-b733-d629fd32b282

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Chainlink/BenqiChainlinkOracle.sol

Report for Chainlink/BenqiChainlinkOracle.sol
https://dashboard.mythx.io/#/console/analyses/27f7ec09-a80a-4bae-a92b-e9f9ecc94c38

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

BaseJumpRateModelV2.sol

Report for BaseJumpRateModelV2.sol
https://dashboard.mythx.io/#/console/analyses/8eb67782-5d77-41fc-b979-44e00f510c00

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
115	(SWC-000) Unknown	Medium	Function could be marked as external.

Governance/Benqi.sol

Report for Governance/Benqi.sol
https://dashboard.mythx.io/#/console/analyses/ba61fc34-dde6-4f9a-9acf-0c665ffc8c63

Line	SWC Title	Severity	Short Description
147	(SWC-000) Unknown	Medium	Function could be marked as external.
160	(SWC-000) Unknown	Medium	Function could be marked as external.
188	(SWC-000) Unknown	Medium	Function could be marked as external.
189	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
262	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

Governance/GovernorAlpha.sol

Report for contracts/Governance/GovernorAlpha.sol
https://dashboard.mythx.io/#/console/analyses/566625a1-f990-42a3-8faa-27715380ec04

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
142	(SWC-000) Unknown	Medium	Function could be marked as external.
143	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
183	(SWC-000) Unknown	Medium	Function could be marked as external.
199	(SWC-000) Unknown	Medium	Function could be marked as external.
209	(SWC-000) Unknown	Medium	Function could be marked as external.
214	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
224	(SWC-000) Unknown	Medium	Function could be marked as external.
229	(SWC-000) Unknown	Medium	Function could be marked as external.
255	(SWC-000) Unknown	Medium	Function could be marked as external.
259	(SWC-000) Unknown	Medium	Function could be marked as external.
272	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
273	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
292	(SWC-000) Unknown	Medium	Function could be marked as external.
297	(SWC-000) Unknown	Medium	Function could be marked as external.
302	(SWC-000) Unknown	Medium	Function could be marked as external.
307	(SWC-000) Unknown	Medium	Function could be marked as external.

Maxmillion.sol

Report for Maximillion.sol
https://dashboard.mythx.io/#/console/analyses/c96c1854-1955-4700-af88-abef3920372e

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
27	(SWC-000) Unknown	Medium	Function could be marked as external.
42	(SWC-134) Message call with hardcoded gas amount	Low	Call with hardcoded gas amount.

PriceOracle.sol

Report for PriceOracle.sol
https://dashboard.mythx.io/#/console/analyses/8acbf09c-4967-45da-b419-143c9bf1195b

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.

QiAvax.sol

Report for QiAvax.sol
https://dashboard.mythx.io/#/console/analyses/c96c1854-1955-4700-af88-abef3920372e

Line	SWC Title	Severity	Short Description
145	(SWC-134) Message call with hardcoded gas amount	Low	Call with hardcoded gas amount.

QiToken.sol

Report for QiToken.sol
https://dashboard.mythx.io/#/console/analyses/ba61fc34-dde6-4f9a-9acf-0c665ffc8c63

Line	SWC Title	Severity	Short Description
25	(SWC-000) Unknown	Medium	Function could be marked as external.
1353	(SWC-000) Unknown	Medium	Function could be marked as external.

QiErc20Delegator.sol

Report for QiErc20Delegator.sol
https://dashboard.mythx.io/#/console/analyses/e15c09c6-930d-42fa-a0fa-2966d89880aa

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
273	(SWC-000) Unknown	Medium	Function could be marked as external.
282	(SWC-000) Unknown	Medium	Function could be marked as external.
292	(SWC-000) Unknown	Medium	Function could be marked as external.
311	(SWC-000) Unknown	Medium	Function could be marked as external.
357	(SWC-000) Unknown	Medium	Function could be marked as external.
408	(SWC-000) Unknown	Medium	Function could be marked as external.

Timelock.sol

Report for Timelock.sol
https://dashboard.mythx.io/#/console/analyses/6e895b2d-94a2-42da-b481-7b03881b2b16

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
36	(SWC-000) Unknown	Medium	Function could be marked as external.
45	(SWC-000) Unknown	Medium	Function could be marked as external.
53	(SWC-000) Unknown	Medium	Function could be marked as external.
60	(SWC-000) Unknown	Medium	Function could be marked as external.
71	(SWC-000) Unknown	Medium	Function could be marked as external.
80	(SWC-000) Unknown	Medium	Function could be marked as external.
95	(SWC-128) DoS With Block Gas Limit	Low	Potentially unbounded data structure passed to builtin.

Unitroller.sol

Report for Unitroller.sol
https://dashboard.mythx.io/#/console/analyses/ba61fc34-dde6-4f9a-9acf-0c665ffc8c63

Line	SWC Title	Severity	Short Description
38	(SWC-000) Unknown	Medium	Function could be marked as external.
58	(SWC-000) Unknown	Medium	Function could be marked as external.
85	(SWC-000) Unknown	Medium	Function could be marked as external.
108	(SWC-000) Unknown	Medium	Function could be marked as external.

WhitePaperInterestRateModel.sol

Report for WhitePaperInterestRateModel.sol
https://dashboard.mythx.io/#/console/analyses/11a69f47-9961-47dc-9284-b7138fa72403

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
79	(SWC-000) Unknown	Medium	Function could be marked as external.

THANK YOU FOR CHOOSING

// HALBORN