HALBORN

Bastion Protocol - EVM contracts

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 20th, 2022 - May 23rd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/17/2022	Michal Bajor
0.2	Document Edits	05/22/2022	Michal Bajor
0.3	Document Edits	05/23/2022	Alpcan Onaran
0.4	Draft Review	05/24/2022	Gabi Urrutia
1.0	Remediation Plan	06/15/2022	Michal Bajor
1.1	Remediation Plan Review	06/16/2022	Gabi Urrutia

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
Alpcan Onaran	Halborn	Alpcan.Onaran@halborn.com

Michal Bajor	Halborn	Michal.Bajor@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Bastion Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on April 20th, 2022 and ending on May 23rd, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Bastion Protocol team.

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 solidity bridge smart contracts. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code 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
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Hardhat)

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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were 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

The review was scoped to the contracts directory using e68e259320e5836cab08697081b2a698d900d8cd commit-id in bastion-protocol/bastion-protocol repository.

- Contracts
  - BSTN.sol
  - Comptroller.sol
  - ComptrollerG1.sol
  - HomoraMath.sol
  - IUniswapV2Pair.sol
  - LockdropVaultV2.sol
  - AggregatorV2V3Interface.sol
  - FluxOracle.sol
  - FluxOracleV1.sol
  - LPOracle.sol
  - NEAROracle.sol
  - StNearFeed.sol
  - StNearFeedV1.sol
  - TwapFeed.sol
  - UQ112x112.sol
  - RewardDistributor.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	4	3

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
IMPROPER ROLE-BASED ACCESS CONTROL POLICY	Medium	FUTURE RELEASE
PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT RECIPIENT'S CONFIRMATION	Medium	SOLVED - 05/25/2022
MISSING DIVISION BY 0 CHECK	Medium	SOLVED - 06/01/2022
USE OF DEPRECATED CHAINLINK API	Low	SOLVED - 05/25/2022
MISSING REENTRANCY GUARD	Low	FUTURE RELEASE
BLOCK TIMESTAMP USAGE IN REWARD CALCULATION	Low	RISK ACCEPTED
MISSING ADDRESS VALIDATION	Low	SOLVED - 05/25/2022
UNNECESSARY REQUIRE STATEMENT IN CONSTRUCTOR	Informational	SOLVED - 05/26/2022
FUNCTIONS CAN BE DECLARED EXTERNAL	Informational	SOLVED - 05/26/2022
USE OF EXPERIMENTAL FEATURES	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPROPER ROLE-BASED ACCESS CONTROL POLICY - MEDIUM

Description:

It was observed that most of the privileged functionality is controlled by the admin. Additional authorization levels are needed to implement the principle of least privilege, also known as least authority, which ensures only authorized processes, users, or programs can access necessary resources or information. Although the contract defines roles responsible for various actions, they can be bypassed by admin.

Code Location:

The owner can access those functions in Comptroller contract:

- _setRewardDistributor
- _setPriceOracle
- _setCloseFactor
- _setCollateralFactor
- _setMaxAssets
- _setLiquidationIncentive
- _supportMarket

The owner can bypass role-based access control of those functions in Comptroller:

- _setMarketBorrowCaps
- _setMintPaused
- _setBorrowPaused
- _setTransferPaused
- _setSeizePaused

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

Removing the admin bypass in functions is advised. In conjunction, using a multi-signature wallet for admin should also be implemented. However, to increase the decentralization of the protocol, it is highly encouraged to implement a governance mechanism. Every significant change should first undergo a voting process and be implemented only if consensus among voters is reached.

Remediation Plan:

PENDING: The Bastion Protocol team confirmed that both the owner and pause guardian are controlled by the team's gnosis and will later be handed to the Timelock contract.

3.2 (HAL-02) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT RECIPIENT'S CONFIRMATION - MEDIUM

Description:

It is observed that the owner can transfer ownership of the contract to a different address. However, such an operation does not require a confirmation from the new owner's address. Incorrect use of such functionality can lead to losing control over contracts, which could not be undone.

Code Location:

Listing 1: contracts/LockdropVaultV2.sol (Line 85)

```
82 function setOwner(address newOwner) external onlyOwner returns (
↳ bool) {
83     require(newOwner != address(0), "DANGER: Attempted to set
↳ owner to 0");
84     require(newOwner != address(this), "DANGER: Attempted to throw
↳  away ownership");
85     owner = newOwner;
86 }
```

Listing 2: contracts/Oracle/FluxOracle.sol (Line 112)

```
110 function setAdmin(address newAdmin) external onlyAdmin() {
111     address oldAdmin = admin;
112     admin = newAdmin;
113
114     emit NewAdmin(oldAdmin, newAdmin);
115 }
```

Listing 3: contracts/Oracle/FluxOracleV1.sol (Line 102)

```
100 function setAdmin(address newAdmin) external onlyAdmin() {
101     address oldAdmin = admin;
102     admin = newAdmin;
103 
104     emit NewAdmin(oldAdmin, newAdmin);
105 }
```

Listing 4: contracts/Oracle/NEAROracle.sol (Line 114)

```
112 function setAdmin(address newAdmin) external onlyAdmin() {
113     address oldAdmin = admin;
114     admin = newAdmin;
115 
116     emit NewAdmin(oldAdmin, newAdmin);
117 }
```

Listing 5: contracts/Oracle/StNearFeedV1.sol (Line 41)

```
 39 function setAdmin(address newAdmin) external onlyAdmin() {
 40     address oldAdmin = admin;
 41     admin = newAdmin;
 42 
 43     emit NewAdmin(oldAdmin, newAdmin);
 44 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to split the ownership transfer functionality into owner setter and acceptOwnership functions. The latter function allows the transfer to be completed by the recipient.

Remediation Plan:

SOLVED: The Bastion Protocol team solved this issue in commit 52f563f88f08f7ee7217fead1d03d8e50b8ccb33 for the FluxOracle contract, which was later renamed to BastionOracle. Furthermore, the Bastion Protocol team confirmed that only the BastionOracle contract will be used from now on, so other contracts reported in this finding are not corrected.

3.3 (HAL-03) MISSING DIVISION BY 0 CHECK - MEDIUM

Description:

The HomoraMath library defines divCeil and fdiv functions, which both implement a division. They are, however, missing the division by 0 check, which will cause an error if encountered.

Code Location:

Listing 6: contracts/HomoraMath.sol (Line 11)

```
10 function divCeil(uint256 lhs, uint256 rhs) internal pure returns (
↳ uint256) {
11     return lhs.add(rhs).sub(1) / rhs;
12 }
```

Listing 7: contracts/HomoraMath.sol (Line 19)

```
18 function fdiv(uint256 lhs, uint256 rhs) internal pure returns (
↳ uint256) {
19     return lhs.mul(2**112) / rhs;
20 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to implement a validation mechanism, which will make sure that division by 0 scenarios are handled properly.

Remediation Plan:

SOLVED: The Bastion Protocol team solved this issue in commit eea975f0489d636687d83fb69534c4c6b31b2e7e

3.4 (HAL-04) USE OF DEPRECATED CHAINLINK API - LOW

Description:

The NEAROracle, LPOracle contracts use Chainlink's deprecated API latestAnswer(). Such functions might suddenly stop working if Chainlink stopped supporting deprecated APIs. This method will return the last value, but it is possible to check if the data is fresh.

Code Location:

Listing 8: contracts/Oracle/NEAROracle.sol (Lines 68,70,76,78)

```
62 function getChainlinkPrice(AggregatorV2V3Interface feed) internal
↳ view returns (uint) {
63     // Chainlink USD-denominated feeds store answers at 8 decimals
64      if (feed.decimals() > 18) {
65         uint decimalDelta = uint(feed.decimals()).sub(uint(18));
66
67         if (decimalDelta > 0) {
68             return uint(feed.latestAnswer()).div(10**decimalDelta)
↳ ;
69         } else {
70             return uint(feed.latestAnswer());
71         }
72     } else {
73         uint decimalDelta = uint(18).sub(uint(feed.decimals()));
74         // Ensure that we don't multiply the result by 0
75         if (decimalDelta > 0) {
76             return uint(feed.latestAnswer()).mul(10**decimalDelta)
↳ ;
77         } else {
78             return uint(feed.latestAnswer());
79         }
80     }
81 }
```

Listing 9: contracts/Oracle/LPOracle.sol (Lines 96,98)

```
87 function getChainlinkPrice(AggregatorV2V3Interface feed)
88     internal
89     view
90     returns (uint256)
91 {
92     // Chainlink USD-denominated feeds store answers at 8 decimals
93     uint256 decimalDelta = uint256(18).sub(feed.decimals());
94     // Ensure that we don't multiply the result by 0
95     if (decimalDelta > 0) {
96         return uint256(feed.latestAnswer()).mul(10**decimalDelta);
97     } else {
98         return uint256(feed.latestAnswer());
99     }
100 }
```

Listing 10: contracts/Oracle/StNearFeed.sol (Lines 28-29)

```
27 function latestAnswer() public view returns (uint) {
28     uint nearUsdPrice = uint(nearUsdFeed.latestAnswer());
29     uint stNearNearPrice = uint(stNearNearFeed.latestAnswer());
30
31     if (feedsDecimals > decimals) {
32         uint decimalsDelta = uint(feedsDecimals).sub(uint(decimals
↳ ));
33         return nearUsdPrice.mul(stNearNearPrice).div(10 **
↳ decimalsDelta);
34     } else {
35         uint decimalsDelta = uint(decimals).sub(uint(feedsDecimals
↳ ));
36         return nearUsdPrice.mul(stNearNearPrice).mul(10 **
↳ decimalsDelta);
37     }
38 }
```

Listing 11: contracts/Oracle/StNearFeedV1.sol (Line 25)

```
24 function latestAnswer() public view returns (uint) {
25     uint nearPrice = uint(nearFeed.latestAnswer());
26     return nearPrice.mul(stNearPrice);
27 }
```

Listing 12: contracts/Oracle/FluxOracle.sol (Lines 66,68,74,76)

```
60 function getChainlinkPrice(AggregatorV2V3Interface feed) internal
↳ view returns (uint) {
61     // Chainlink USD-denominated feeds store answers at 8 decimals
62      if (feed.decimals() > 18) {
63         uint decimalDelta = uint(feed.decimals()).sub(uint(18));
64
65         if (decimalDelta > 0) {
66             return uint(feed.latestAnswer()).div(10**decimalDelta)
↳ ;
67         } else {
68             return uint(feed.latestAnswer());
69         }
70     } else {
71         uint decimalDelta = uint(18).sub(uint(feed.decimals()));
72         // Ensure that we don't multiply the result by 0
73         if (decimalDelta > 0) {
74             return uint(feed.latestAnswer()).mul(10**decimalDelta)
↳ ;
75         } else {
76             return uint(feed.latestAnswer());
77         }
78     }
79 }
```

Listing 13: contracts/Oracle/FluxOracleV1.sol (Lines 65,67)

```
60 function getChainlinkPrice(AggregatorV2V3Interface feed) internal
↳ view returns (uint) {
61     // Chainlink USD-denominated feeds store answers at 8 decimals
62     uint decimalDelta = uint(18).sub(feed.decimals());
63     // Ensure that we don't multiply the result by 0
64     if (decimalDelta > 0) {
65         return uint(feed.latestAnswer()).mul(10**decimalDelta);
66     } else {
67         return uint(feed.latestAnswer());
68     }
69 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to use latestRoundData() method instead of latestAnswer(). This method allows executing some extra validations as shown as below:

Listing 14: Extra Validations (Lines 2,3,4)

```
1 (roundId, rawPrice, , updateTime, answeredInRound) =
↳ AggregatorV3Interface(feed).latestRoundData();
2         require(rawPrice > 0, "Chainlink price cannot be lower
↳ than 0");
3         require(updateTime != 0, "Round is in incompleted state");
4         require(answeredInRound >= roundId, "Stale price");
```

Remediation Plan:

SOLVED: The Bastion Protocol solved this issue in commit fbf52f880bfec7790194c7bd16d610cc8c17f9cb for the FluxOracle contract, which was later renamed to BastionOracle. Furthermore, the Bastion Protocol team confirmed that only the BastionOracle contract will be used from now on, so other contracts reported in this finding are not corrected.

3.5 (HAL-05) MISSING REENTRANCY GUARD - LOW

Description:

During tests, it is observed that claimReward function updates rewardAccrued[] variable after contacting an external address using transfer(), transfer() function is not prone to re-entrancy attacks, however this function maybe at risk if this functionality chance. Therefore, to protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

Code Location:

Listing 15: contracts/RewardDistributor.sol (Lines 276,291)

```
525     function claimReward(
526         uint8 rewardType,
527         address payable[] memory holders,
528         CToken[] memory cTokens,
529         bool borrowers,
530         bool suppliers
531     ) public payable {
532         require(rewardType < rewardAddresses.length, "rewardType
 ↳ is invalid");
533         for (uint256 i = 0; i < cTokens.length; i++) {
534             CToken cToken = cTokens[i];
535             require(
536                 comptroller.isMarketListed(address(cToken)),
537                 "market must be listed"
538             );
539             if (borrowers == true) {
540                 Exp memory borrowIndex = Exp({mantissa: cToken.
 ↳ borrowIndex()});
541                 updateRewardBorrowIndex(
```

```
542                     rewardType,
543                     address(cToken),
544                     borrowIndex
545                 );
546
547                 for (uint256 j = 0; j < holders.length; j++) {
548                     distributeBorrowerReward(
549                         rewardType,
550                         address(cToken),
551                         holders[j],
552                         borrowIndex
553                     );
554
555                     rewardAccrued[rewardType][holders[j]] =
 ↳ grantRewardInternal(
556                         rewardType,
557                         holders[j],
558                         rewardAccrued[rewardType][holders[j]]
559                     );
560                 }
561             }
562
563             if (suppliers == true) {
564                 updateRewardSupplyIndex(rewardType, address(cToken
 ↳ ));
565                 for (uint256 j = 0; j < holders.length; j++) {
566                     distributeSupplierReward(
567                         rewardType,
568                         address(cToken),
569                         holders[j]
570                     );
571
572                     rewardAccrued[rewardType][holders[j]] =
 ↳ grantRewardInternal(
573                         rewardType,
574                         holders[j],
575                         rewardAccrued[rewardType][holders[j]]
576                     );
577                 }
578             }
579         }
```

Listing 16: contracts/RewardDistributor.sol (Line 312)

```
586
587     function grantRewardInternal(uint8 rewardType, address payable
 ↳  user,  uint256 amount ) internal returns (uint256) {
588
589         address rewardAddress = rewardAddresses[rewardType];
590         EIP20Interface reward = EIP20Interface(rewardAddress);
591         uint256 rewardRemaining = reward.balanceOf(address(this));
592         if (amount > 0 && amount <= rewardRemaining) {
593
594             reward.transfer(user, amount);
595
596             return 0;
597         }
598
599         return amount;
600     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

The functions on the code location section have missing nonReentrant modifiers. It is recommended to add OpenZeppelin ReentrancyGuard library to the project and use the nonReentrant modifier to avoid introducing future re-entrancy vulnerabilities.

Remediation Plan:

PENDING: The Bastion Protocol team will deploy a fix in the new BoostRewardDistributor in bastion-dao.

3.6 (HAL-06) BLOCK TIMESTAMP USAGE IN REWARD CALCULATION - LOW

Description:

During manual reviews, the usage of block.timestamp as a part of a financial mechanism was noticed. block.timestamp can be influenced by miners to a certain degree. There is a risk associated with the miners colluding on time manipulation to influence those mechanisms. However, as the contracts audited here are designed to be deployed on Aurora EVM, which has significantly reduced block times compared to Ethereum, this findings' severity is reduced.

Code Location:

Listing 17: contracts/Oracle/TwapFeed.sol (Line 121)

```
115 function update()
116         external
117         returns (uint224 price, uint32 T)
118     {
119         require(pair.initialized, "TwapFeed: NOT_INITIALIZED");
120 
121         uint32 blockTimestamp = getBlockTimestamp();
122         uint32 lastUpdateTimestamp =
123             pair.latestIsSlotA ? pair.lastUpdateSlotA : pair.
 ↳ lastUpdateSlotB;
124         uint256 priceCumulativeCurrent =
125             getPriceCumulativeCurrent(pair.asToken0);
126         uint256 priceCumulativeLast;
127 
128         if (blockTimestamp - lastUpdateTimestamp >= MIN_T) {
129             // update price
130             priceCumulativeLast = pair.latestIsSlotA
131                 ? pair.priceCumulativeSlotA
132                 : pair.priceCumulativeSlotB;
133             if (pair.latestIsSlotA) {
134                 pair.priceCumulativeSlotB = priceCumulativeCurrent
 ↳ ;
135                 pair.lastUpdateSlotB = blockTimestamp;
```

```
136             } else {
137                 pair.priceCumulativeSlotA = priceCumulativeCurrent
 ↳ ;
138                 pair.lastUpdateSlotA = blockTimestamp;
139             }
140             pair.latestIsSlotA = !pair.latestIsSlotA;
141             emit PriceUpdate(
142                 uniswapV2Pair,
143                 priceCumulativeCurrent,
144                 blockTimestamp,
145                 !pair.latestIsSlotA
146             );
147         } else {
148             // don't update; return price using previous
 ↳ priceCumulative
149             lastUpdateTimestamp = pair.latestIsSlotA
150                 ? pair.lastUpdateSlotB
151                 : pair.lastUpdateSlotA;
152             priceCumulativeLast = pair.latestIsSlotA
153                 ? pair.priceCumulativeSlotB
154                 : pair.priceCumulativeSlotA;
155         }
156
157         T = blockTimestamp - lastUpdateTimestamp; // overflow is
 ↳ desired
158         require(T >= MIN_T, "TwapFeed: NOT_READY"); //reverts only
 ↳  if the pair has just been initialized
159         // / is safe, and - overflow is desired
160         price = toUint224((priceCumulativeCurrent -
 ↳ priceCumulativeLast) / T);
161     }
```

Listing 18: contracts/RewardDistributor.sol (Lines 182,218)

```
153 function setRewardSpeedInternal(
154     uint8 rewardType,
155     CToken cToken,
156     uint256 newSupplySpeed,
157     uint256 newBorrowSpeed
158 ) internal {
159     // Handle new supply speeed
160     uint256 currentRewardSupplySpeed = rewardSupplySpeeds[
 ↳ rewardType][
161         address(cToken)
```

```
162     ];
163     if (currentRewardSupplySpeed != 0) {
164         // note that JOE speed could be set to 0 to halt liquidity
 ↳  rewards for a market
165         updateRewardSupplyIndex(rewardType, address(cToken));
166     } else if (newSupplySpeed != 0) {
167         // Add the JOE market
168         require(
169             comptroller.isMarketListed(address(cToken)),
170             "reward market is not listed"
171         );
172 
173         if (
174             rewardSupplyState[rewardType][address(cToken)].index
 ↳ == 0 &&
175             rewardSupplyState[rewardType][address(cToken)].
 ↳ timestamp == 0
176         ) {
177             rewardSupplyState[rewardType][
178                 address(cToken)
179             ] = RewardMarketState({
180                 index: rewardInitialIndex,
181                 timestamp: safe32(
182                     getBlockTimestamp(),
183                     "block timestamp exceeds 32 bits"
184                 )
185             });
186         }
187     }
188 
189     if (currentRewardSupplySpeed != newSupplySpeed) {
190         rewardSupplySpeeds[rewardType][address(cToken)] =
 ↳ newSupplySpeed;
191         emit RewardSupplySpeedUpdated(rewardType, cToken,
 ↳ newSupplySpeed);
192     }
193 
194     // Handle new borrow speed
195     uint256 currentRewardBorrowSpeed = rewardBorrowSpeeds[
 ↳ rewardType][
196         address(cToken)
197     ];
198     if (currentRewardBorrowSpeed != 0) {
```

```
199         // note that JOE speed could be set to 0 to halt liquidity
 ↳  rewards for a market
200         Exp memory borrowIndex = Exp({mantissa: cToken.borrowIndex
 ↳ ()});
201         updateRewardBorrowIndex(rewardType, address(cToken),
 ↳ borrowIndex);
202     } else if (newBorrowSpeed != 0) {
203         // Add the JOE market
204         require(
205             comptroller.isMarketListed(address(cToken)),
206             "reward market is not listed"
207         );
208 
209         if (
210             rewardBorrowState[rewardType][address(cToken)].index
 ↳ == 0 &&
211             rewardBorrowState[rewardType][address(cToken)].
 ↳ timestamp == 0
212         ) {
213             rewardBorrowState[rewardType][
214                 address(cToken)
215             ] = RewardMarketState({
216                 index: rewardInitialIndex,
217                 timestamp: safe32(
218                     getBlockTimestamp(),
219                     "block timestamp exceeds 32 bits"
220                 )
221             });
222         }
223     }
224 
225     if (currentRewardBorrowSpeed != newBorrowSpeed) {
226         rewardBorrowSpeeds[rewardType][address(cToken)] =
 ↳ newBorrowSpeed;
227         emit RewardBorrowSpeedUpdated(rewardType, cToken,
 ↳ newBorrowSpeed);
228     }
229 }
```

Listing 19: contracts/RewardDistributor.sol (Line 244)

```
236 function updateRewardSupplyIndex(uint8 rewardType, address cToken)
237     internal
238 {
```

```
239     require(rewardType < rewardAddresses.length, "rewardType is
 ↳ invalid");
240     RewardMarketState storage supplyState = rewardSupplyState[
 ↳ rewardType][
241         cToken
242     ];
243     uint256 supplySpeed = rewardSupplySpeeds[rewardType][cToken];
244     uint256 blockTimestamp = getBlockTimestamp();
245     uint256 deltaTimestamps = sub_(
246         blockTimestamp,
247         uint256(supplyState.timestamp)
248     );
249     if (deltaTimestamps > 0 && supplySpeed > 0) {
250         uint256 supplyTokens = CToken(cToken).totalSupply();
251         uint256 rewardAccrued = mul_(deltaTimestamps, supplySpeed)
 ↳ ;
252         Double memory ratio = supplyTokens > 0
253             ? fraction(rewardAccrued, supplyTokens)
254             : Double({mantissa: 0});
255         Double memory index = add_(
256             Double({mantissa: supplyState.index}),
257             ratio
258         );
259         rewardSupplyState[rewardType][cToken] = RewardMarketState
 ↳ ({
260             index: safe224(index.mantissa, "new index exceeds 224
 ↳ bits"),
261             timestamp: safe32(
262                 blockTimestamp,
263                 "block timestamp exceeds 32 bits"
264             )
265         });
266     } else if (deltaTimestamps > 0) {
267         supplyState.timestamp = safe32(
268             blockTimestamp,
269             "block timestamp exceeds 32 bits"
270         );
271     }
272 }
```

Listing 20: contracts/RewardDistributor.sol (Line 290)

```
280 function updateRewardBorrowIndex(
281     uint8 rewardType,
282     address cToken,
283     Exp memory marketBorrowIndex
284 ) internal {
285     require(rewardType < rewardAddresses.length, "rewardType is
 ↳ invalid");
286     RewardMarketState storage borrowState = rewardBorrowState[
 ↳ rewardType][
287         cToken
288     ];
289     uint256 borrowSpeed = rewardBorrowSpeeds[rewardType][cToken];
290     uint256 blockTimestamp = getBlockTimestamp();
291     uint256 deltaTimestamps = sub_(
292         blockTimestamp,
293         uint256(borrowState.timestamp)
294     );
295     if (deltaTimestamps > 0 && borrowSpeed > 0) {
296         uint256 borrowAmount = div_(
297             CToken(cToken).totalBorrows(),
298             marketBorrowIndex
299         );
300         uint256 rewardAccrued = mul_(deltaTimestamps, borrowSpeed)
 ↳ ;
301         Double memory ratio = borrowAmount > 0
302             ? fraction(rewardAccrued, borrowAmount)
303             : Double({mantissa: 0});
304         Double memory index = add_(
305             Double({mantissa: borrowState.index}),
306             ratio
307         );
308         rewardBorrowState[rewardType][cToken] = RewardMarketState
 ↳ ({
309             index: safe224(index.mantissa, "new index exceeds 224
 ↳ bits"),
310             timestamp: safe32(
311                 blockTimestamp,
312                 "block timestamp exceeds 32 bits"
313             )
314         });
315     } else if (deltaTimestamps > 0) {
316         borrowState.timestamp = safe32(
317             blockTimestamp,
```

```
318             "block timestamp exceeds 32 bits"
319         );
320     }
321 }
```

Risk Level:

Likelihood - 2
Impact - 2

Proof Of Concept:

Listing 21

```
 1 import { expect } from "./chai-setup";
 2
 3 import { ethers } from "hardhat";
 4 import { parseEther, parseUnits } from "ethers/lib/utils";
 5 import { BigNumber } from "ethers";
 6 import {
 7  CErc20,
 8  Comptroller,
 9  FaucetToken,
10  RewardDistributor,
11 } from "../typechain-types";
12 import {
13  makeCErc20,
14  makeComptroller,
15  makeRewardDistributor,
16  makeToken,
17 } from "./common/Compound";
18 import { SignerWithAddress } from "@nomiclabs/hardhat-ethers/
↳ signers";
19
20 describe("BSTN Liquidity Mining", function () {
21  let alice: SignerWithAddress;
22  let malicious_miner: SignerWithAddress;
23
24  let rewardDistributor: RewardDistributor;
25  let comptroller: Comptroller;
26  let BSTN: FaucetToken;
```

```
27  let NEAR: FaucetToken;
28  let cNEAR: CErc20;
29  before(async function () {
30    [, alice, malicious_miner] = await ethers.getSigners();
31
32    BSTN = await makeToken({
33      name: "Bastion",
34      symbol: "BSTN",
35      decimals: 18,
36    });
37    rewardDistributor = await makeRewardDistributor();
38
39    await BSTN.mint(rewardDistributor.address, parseUnits("4000000"
↳ , 18));
40
41    comptroller = await makeComptroller({
42      rewardDistributor: rewardDistributor.address,
43    });
44
45    NEAR = await makeToken({
46      name: "NEAR",
47      symbol: "NEAR",
48      decimals: 24,
49    });
50    cNEAR = await makeCErc20({
51      comptroller,
52      name: "Bastion NEAR",
53      symbol: "cNEAR",
54      underlying: NEAR,
55      supportMarket: true,
56      collateralFactor: parseEther("0.4"),
57    });
58
59    await rewardDistributor.addRewardAddress(BSTN.address);
60
61    await rewardDistributor._setRewardSpeed(
62      0,
63      cNEAR.address,
64      0,
65      parseEther("0.01")
66    );
67
68  });
69
```

```
70  const ALICE_DEPOSIT_AMOUNT = "1";
71  const ALICE_BORROW_AMOUNT = "0.4";
72  it("BSTN Distribution Without Timestamp Manipulation", async ()
↳ => {
73
74
75
76
77    const underlyingDecimals = await NEAR.decimals();
78    await NEAR.mint(
79      alice.address,
80      parseUnits(ALICE_DEPOSIT_AMOUNT, underlyingDecimals)
81    );
82
83    await NEAR.connect(alice).approve(
84      cNEAR.address,
85      ethers.constants.MaxUint256
86    );
87    await cNEAR
88      .connect(alice)
89      .mint(parseUnits(ALICE_DEPOSIT_AMOUNT, underlyingDecimals));
90    await cNEAR
91      .connect(alice)
92      .borrow(parseUnits(ALICE_BORROW_AMOUNT, underlyingDecimals));
93    console.log("");
94    console.log("Normal user (Alice) borrows 0.4 and deposits 1");
95    await ethers.provider.send("evm_increaseTime", [10000]);
96
97    console.log("Time passes evm_increaseTime, [10000]");
98
99    await rewardDistributor
100     .connect(alice)
101     ["claimReward(uint8,address)"](0, alice.address);
102
103   const BSTNBalance = await BSTN.balanceOf(alice.address);
104   console.log("Alice claims rewards")
105   console.log(BSTNBalance);
106   expect(BSTNBalance).to.be.gt(BigNumber.from(0));
107 });
108
109
110
111 });
112
```

```
113 describe("BSTN Liquidity Mining (Manipulated timestamp)", function
 ↳  () {
114   let alice: SignerWithAddress;
115   let malicious_miner: SignerWithAddress;
116
117   let rewardDistributor: RewardDistributor;
118   let comptroller: Comptroller;
119   let BSTN: FaucetToken;
120   let NEAR: FaucetToken;
121   let cNEAR: CErc20;
122   before(async function () {
123     [, alice, malicious_miner] = await ethers.getSigners();
124
125     BSTN = await makeToken({
126       name: "Bastion",
127       symbol: "BSTN",
128       decimals: 18,
129     });
130     rewardDistributor = await makeRewardDistributor();
131
132     await BSTN.mint(rewardDistributor.address, parseUnits("4000000"
 ↳ , 18));
133
134     comptroller = await makeComptroller({
135       rewardDistributor: rewardDistributor.address,
136     });
137
138     NEAR = await makeToken({
139       name: "NEAR",
140       symbol: "NEAR",
141       decimals: 24,
142     });
143     cNEAR = await makeCErc20({
144       comptroller,
145       name: "Bastion NEAR",
146       symbol: "cNEAR",
147       underlying: NEAR,
148       supportMarket: true,
149       collateralFactor: parseEther("0.4"),
150     });
151
152     await rewardDistributor.addRewardAddress(BSTN.address);
153
154     await rewardDistributor._setRewardSpeed(
```

```
155       0,
156       cNEAR.address,
157       0,
158       parseEther("0.01")
159     );
160
161   });
162
163   const ALICE_DEPOSIT_AMOUNT = "1";
164   const ALICE_BORROW_AMOUNT = "0.4";
165   it("BSTN Distribution With Timestamp Manipulation", async () => {
166
167     await rewardDistributor._setRewardSpeed(
168       0,
169       cNEAR.address,
170       0,
171       parseEther("0.01")
172     );
173
174
175     const underlyingDecimals = await NEAR.decimals();
176     await NEAR.mint(
177       malicious_miner.address,
178       parseUnits(ALICE_DEPOSIT_AMOUNT, underlyingDecimals)
179     );
180
181     await NEAR.connect(malicious_miner).approve(
182       cNEAR.address,
183       ethers.constants.MaxUint256
184     );
185     await cNEAR
186       .connect(malicious_miner)
187       .mint(parseUnits(ALICE_DEPOSIT_AMOUNT, underlyingDecimals));
188     await cNEAR
189       .connect(malicious_miner)
190       .borrow(parseUnits(ALICE_BORROW_AMOUNT, underlyingDecimals));
191       console.log("");
192       console.log("Malicious user (Miner) borrows 0.4 and deposits
 ↳ 1");
193       await ethers.provider.send("evm_increaseTime", [10015]);
194        console.log("Time passes evm_increaseTime, [10015],
 ↳ Malicious miner manipulates block timestamp +15");
195
196
```

```
197     await rewardDistributor
198       .connect(malicious_miner)
199       ["claimReward(uint8,address)"](0, malicious_miner.address);
200
201     const BSTNBalance = await BSTN.balanceOf(malicious_miner.
 ↳ address);
202
203     console.log("Malicious miner claims rewards")
204     console.log(BSTNBalance);
205     expect(BSTNBalance).to.be.gt(BigNumber.from(0));
206   });
207
208 });
209
```

```
Normal user (Alice) borrows 0.4 and deposits 1
Time passes evm_increaseTime, [10000]
Alice claims rewards
BigNumber { value: "10002000000000000000" }
    ✓ BSTN Distribution Without Timestamp Manipulation (1130ms)

  BSTN Liquidity Mining (Manipulated timestamp)
Duplicate definition of ActionPaused (ActionPaused(string,bool), ActionPaused(address,string,bool))
Duplicate definition of ActionPaused (ActionPaused(string,bool), ActionPaused(address,string,bool))

Malicious user (Miner) borrows 0.4 and deposits 1
Time passes evm_increaseTime, [10015], Malicious miner manipulates block timestamp +15
Malicious miner claims rewards
BigNumber { value: "10017000000000000000" }
    ✓ BSTN Distribution With Timestamp Manipulation (1037ms)


  2 passing (5s)
```

Recommendation:

It is recommended to use block.number instead of block.timestamp to reduce the risk of MEV attacks.

Remediation Plan:

RISK ACCEPTED: The Bastion Protocol team accepted the risk of this finding.

3.7 (HAL-07) MISSING ADDRESS VALIDATION - LOW

Description:

Multiple contracts are missing a safety check inside their constructors and multiple functions. Setters of address type parameters should include a zero-address check. Otherwise, contracts' functionalities may become inaccessible or tokens are burnt forever.

Code Location:

Listing 22: contracts/LockdropVaultV2.sol (Line 34)

```
28 constructor(string memory name_,
29             address ctoken_,
30             uint256 claimUnlockTime_) public {
31
32     require(claimUnlockTime_ > now, "claim unlock time is before
↳ current time");
33     name = name_;
34     ctoken = ctoken_;
35     claimUnlockTime = claimUnlockTime_;
36     owner = msg.sender;
37 }
```

Listing 23: contracts/Oracle/FluxOracle.sol (Line 112)

```
110 function setAdmin(address newAdmin) external onlyAdmin() {
111     address oldAdmin = admin;
112     admin = newAdmin;
113
114     emit NewAdmin(oldAdmin, newAdmin);
115 }
```

Listing 24: contracts/Oracle/FluxOracleV1.sol (Line 102)

```
100 function setAdmin(address newAdmin) external onlyAdmin() {
101     address oldAdmin = admin;
102     admin = newAdmin;
103
104     emit NewAdmin(oldAdmin, newAdmin);
105 }
```

Listing 25: contracts/Oracle/NEAROracle.sol (Line 114)

```
112 function setAdmin(address newAdmin) external onlyAdmin() {
113     address oldAdmin = admin;
114     admin = newAdmin;
115
116     emit NewAdmin(oldAdmin, newAdmin);
117 }
```

Listing 26: contracts/Oracle/StNearFeed.sol (Lines 17,18)

```
16 constructor (address _nearUsdFeed, address _stNearNearFeed) public
↳  {
17     nearUsdFeed = AggregatorV2V3Interface(_nearUsdFeed);
18     stNearNearFeed = AggregatorV2V3Interface(_stNearNearFeed);
19
20     uint8 nearUsdDecimals = nearUsdFeed.decimals();
21     uint8 stNearNearDecimals = stNearNearFeed.decimals();
22     feedsDecimals = nearUsdDecimals + stNearNearDecimals;
23     require(feedsDecimals >= nearUsdDecimals && feedsDecimals >=
↳ stNearNearDecimals, "overflow");
24 }
```

Listing 27: contracts/Oracle/StNearFeedV1.sol (Line 19)

```
17 constructor (uint _stNearPrice, address _nearFeed) public {
18     admin = msg.sender;
19     nearFeed = AggregatorV2V3Interface(_nearFeed);
20     stNearPrice = _stNearPrice;
21 }
```

Listing 28: contracts/Oracle/StNearFeedV1.sol (Line 30)

```
29 function setNearFeed(address _nearFeed) public onlyAdmin {
30     nearFeed = AggregatorV2V3Interface(_nearFeed);
31 }
```

Listing 29: contracts/Oracle/StNearFeedV1.sol (Line 41)

```
39 function setAdmin(address newAdmin) external onlyAdmin() {
40     address oldAdmin = admin;
41     admin = newAdmin;
42
43     emit NewAdmin(oldAdmin, newAdmin);
44 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

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

Remediation Plan:

SOLVED: The Bastion Protocol team solved this issue in commit 52f563f88f08f7ee7217fead1d03d8e50b8ccb33 for the FluxOracle contract, which was later renamed to BastionOracle. Furthermore, the Bastion Protocol team confirmed that only the BastionOracle contract will be used from now on, so other contracts reported in this finding are not corrected.

3.8 (HAL-08) UNNECESSARY REQUIRE STATEMENT IN CONSTRUCTOR - INFORMATIONAL

Description:

The TwapFeed contract's constructor contains a require statement, verifying if the Pair struct stored in storage was initialized. Such validation is unnecessary in the constructor, as the storage is guaranteed not to be initialized yet. As a consequence, this comparison increases the execution cost without any benefits.

Code Location:

Listing 30: contracts/Oracle/TwapFeed.sol (Lines 47-50)

```
39   constructor (address _uniswapV2Pair, bool asToken0) public {
40       uniswapV2Pair = _uniswapV2Pair;
41          pair.asToken0 = asToken0;
42
43 EIP20Interface token0 = EIP20Interface(IUniswapV2Pair(
↳ uniswapV2Pair).token0());
44 EIP20Interface token1 = EIP20Interface(IUniswapV2Pair(
↳ uniswapV2Pair).token1());
45       pairDecimals = asToken0 ? 18 + token1.decimals() - token0.
↳ decimals() : 18 + token0.decimals() - token1.decimals();
46
47       require(
48           !pair.initialized,
49           "TwapFeed: ALREADY_INITIALIZED"
50       );
51
52       uint256 priceCumulativeCurrent =
53           getPriceCumulativeCurrent(pair.asToken0);
54
55       uint32 blockTimestamp = getBlockTimestamp();
56       pair.priceCumulativeSlotA = priceCumulativeCurrent;
57       pair.priceCumulativeSlotB = priceCumulativeCurrent;
58       pair.lastUpdateSlotA = blockTimestamp;
```

```
59        pair.lastUpdateSlotB = blockTimestamp;
60        pair.latestIsSlotA = true;
61        pair.initialized = true;
62        emit PriceUpdate(
63            uniswapV2Pair,
64            priceCumulativeCurrent,
65            blockTimestamp,
66            true
67        );
68    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Delete the unnecessary require statements.

Remediation Plan:

SOLVED: The Bastion Protocol team solved this issue in commit 851e5e1aca4b4d01b4f007082604e6e9a5c8be85

3.9 (HAL-09) FUNCTIONS CAN BE DECLARED EXTERNAL - INFORMATIONAL

Description:

Multiple functions are declared as public. However, they do not appear to be called from within the contract in which they are defined. Suppose a function is designed to be called by users and is not intended to be accessible internally to other functions. In that case, it is better to declare them as external to reduce the gas cost associated with their execution.

Code Location:

Following is the list of functions that can be declared as external.

FluxOracle.sol:

- getUnderlyingPrice

FluxOracleV1.sol:

- getUnderlyingPrice

LPOracle.sol:

- getUnderlyingPrice

NEAROracle.sol:

- getUnderlyingPrice

StNearFeed.sol:

- latestAnswer

StNearFeedV1.sol:

- setStNearPrice
- setNearFeed

Comptroller.sol and ComptrollerG1.sol:

- enterMarkets
- getAccountLiquidity
- getHypotheticalAccountLiquidity
- _setRewardDistributor
- _setPriceOracle
- _setPauseGuardian
- _setMintPaused
- _setBorrowPaused
- _setTransferPaused
- _setSeizePaused
- _become
- getAllMarkets
- isMarketListed

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Declare functions designed to be called externally as external.

Remediation Plan:

SOLVED: The Bastion Protocol team solved this issue in commit 5fae05ea94bf13bedb4f80155adaa6c2ab97b9a3

3.10 (HAL-10) USE OF EXPERIMENTAL FEATURES - INFORMATIONAL

Description:

Comptroller, ComptrollerG1 and RewardDistributor contracts use ABIEncoderV2 experimental feature. Experimental features are not part of the official release version. They are not extensively tested, and their usage may result in unexpected behavior.

Code Location:

Listing 31: contracts/Comptroller.sol,ComptrollerG1.sol,RewardDistributor.sol (Line 2)

```
2 pragma experimental ABIEncoderV2;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not use experimental features in production environments. If possible, the contracts should be rewritten not to utilize the ABIEncoderV2.

Remediation Plan:

ACKNOWLEDGED: The Bastion Protocol team acknowledged this finding.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. 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 code-base.

Slither results:

To reduce the report size, Informational and Optimization findings reported by Slither were omitted.

Impact	Short Description	Contract
High	RewardDistributor (contracts/RewardDistributor.sol#60-662) contract sets array length with a user-controlled value: - rewardAddresses.push(newRewardAddress) (contracts/RewardDistributor.sol#606)	Comptroller
High	Comptroller (contracts/Comptroller.sol#16-1553) contract sets array length with a user-controlled value: - allMarkets.push(CToken(cToken)) (contracts/Comptroller.sol#1340)	Comptroller
High	Comptroller (contracts/Comptroller.sol#16-1553) contract sets array length with a user-controlled value: - accountAssets[borrower].push(cToken) (contracts/Comptroller.sol#229)	Comptroller
High	RewardDistributor.grantRewardInternal(uint8, address,uint256) (contracts/RewardDistributor.sol#565-579) ignores return value by reward.transfer(user,amount) (contracts/RewardDistributor.sol#574)	Comptroller
High	UnitrollerAdminStorage.comptrollerImplementation (contracts/ComptrollerStorage.sol#20) is never initialized. It is used in: - Comptroller.adminOrInitializing() (contracts/Comptroller.sol#1516-1518)	Comptroller
High	TwapFeed.getBlockTimestamp() (contracts/Oracle/TwapFeed.sol#164-166) uses a weak PRNG: "uint32(block.timestamp % 2 ** 32) (contracts/Oracle/TwapFeed.sol#165)"	TwapFeed
High	RewardDistributor (contracts/RewardDistributor.sol#60-662) contract sets array length with a user-controlled value: - rewardAddresses.push(newRewardAddress) (contracts/RewardDistributor.sol#606)	RewardDistributor
High	Comptroller (contracts/Comptroller.sol#16-1553) contract sets array length with a user-controlled value: - allMarkets.push(CToken(cToken)) (contracts/Comptroller.sol#1340)	RewardDistributor

Impact	Short Description	Contract
High	Comptroller (contracts/Comptroller.sol#16-1553) contract sets array length with a user-controlled value: - accountAssets[borrower].push(cToken) (contracts/Comptroller.sol#229)	RewardDistributor
High	RewardDistributor.grantRewardInternal(uint8, address,uint256) (contracts/RewardDistributor.sol#565-579) ignores return value by reward.transfer(user,amount) (contracts/RewardDistributor.sol#574)	RewardDistributor
High	UnitrollerAdminStorage.comptrollerImplementation (contracts/ComptrollerStorage.sol#20) is never initialized. It is used in: - Comptroller.adminOrInitializing() (contracts/Comptroller.sol#1516-1518)	RewardDistributor
High	ComptrollerG1 (contracts/ComptrollerG1.sol#16-1526) contract sets array length with a user-controlled value: - accountAssets[borrower].push(cToken) (contracts/ComptrollerG1.sol#221)	ComptrollerG1
High	ComptrollerG1 (contracts/ComptrollerG1.sol#16-1526) contract sets array length with a user-controlled value: - allMarkets.push(CToken(cToken)) (contracts/ComptrollerG1.sol#1313)	ComptrollerG1
High	RewardDistributor (contracts/RewardDistributor.sol#60-662) contract sets array length with a user-controlled value: - rewardAddresses.push(newRewardAddress) (contracts/RewardDistributor.sol#606)	ComptrollerG1
High	Comptroller (contracts/Comptroller.sol#16-1553) contract sets array length with a user-controlled value: - allMarkets.push(CToken(cToken)) (contracts/Comptroller.sol#1340)	ComptrollerG1
High	Comptroller (contracts/Comptroller.sol#16-1553) contract sets array length with a user-controlled value: - accountAssets[borrower].push(cToken) (contracts/Comptroller.sol#229)	ComptrollerG1

Impact	Short Description	Contract
High	RewardDistributor.grantRewardInternal(uint8, address,uint256) (contracts/RewardDistributor.sol#565-579) ignores return value by reward.transfer(user,amount) (contracts/RewardDistributor.sol#574)	ComptrollerG1
High	UnitrollerAdminStorage.comptrollerImplementation (contracts/ComptrollerStorage.sol#20) is never initialized. It is used in: - ComptrollerG1.adminOrInitializing() (contracts/ComptrollerG1.sol#1489-1491)	ComptrollerG1
High	UnitrollerAdminStorage.comptrollerImplementation (contracts/ComptrollerStorage.sol#20) is never initialized. It is used in: - Comptroller.adminOrInitializing() (contracts/Comptroller.sol#1516-1518)	ComptrollerG1

Impact	Short Description	Contract
Medium	Comptroller.transferVerify(address,address, address,uint256) (contracts/Comptroller.sol#814-830) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#827)	Comptroller
Medium	Comptroller.repayBorrowVerify(address,address, address,uint256,uint256) (contracts/Comptroller.sol#586-604) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#601)	Comptroller
Medium	Comptroller.liquidateBorrowVerify(address, address,address,address,uint256,uint256) (contracts/Comptroller.sol#674-694) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#691)	Comptroller
Medium	Comptroller.borrowVerify(address,address,uint256) (contracts/Comptroller.sol#528-542) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#539)	Comptroller
Medium	Comptroller.mintVerify(address,address,uint256, uint256) (contracts/Comptroller.sol#348-364) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#361)	Comptroller
Medium	Comptroller.seizeVerify(address,address,address, address,uint256) (contracts/Comptroller.sol#754-772) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#769)	Comptroller
Medium	RewardDistributor.setRewardSpeedInternal(uint8, CToken,uint256,uint256) (contracts/RewardDistributor.sol#153-229) uses a dangerous strict equality: -rewardSupplyState[rewardType][address(cToken)].index == 0 && rewardSupplyState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol#174-175)	Comptroller

Impact	Short Description	Contract
Medium	RewardDistributor.setRewardSpeedInternal(uint8, CToken,uint256,uint256) (contracts/RewardDistributor.sol#153-229) uses a dangerous strict equality: - rewardBorrowState[rewardType][address(cToken)].index == 0 && rewardBorrowState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol#210-211)	Comptroller
Medium	Contract locking ether found: Contract RewardDistributor (contracts/RewardDistributor.sol#60-662) has payable functions: - RewardDistributor.claimReward(uint8,address[],CToken[],bool,bool) (contracts/RewardDistributor.sol#504-555) - RewardDistributor.fallback() (contracts/RewardDistributor.sol#657) But does not have a function to withdraw the ether	Comptroller
Medium	Comptroller.borrowAllowed(address,address, uint256).err_scope_0 (contracts/Comptroller.sol#494) is a local variable never initialized	Comptroller
Medium	Comptroller._supportMarket(CToken) (contracts/Comptroller.sol#1302-1334) ignores return value by cToken.isCToken() (contracts/Comptroller.sol#1319)	Comptroller
Medium	LPOracle.getLPPrice(address) (contracts/Oracle/LPOracle.sol#41-53) performs a multiplication on the result of a division: -sqrtK.mul(2).mul(HomoraMath.sqrt(px0)).div(2 ** 56).mul(HomoraMath.sqrt(px1)).div(2 ** 56) (contracts/Oracle/LPOracle.sol#52)	LPOracle
Medium	LPOracle._setLPs(address[],bool[]) (contracts/Oracle/LPOracle.sol#105-116) ignores return value by IUniswapV2Pair(tokenAddresses[i]).token1() (contracts/Oracle/LPOracle.sol#112)	LPOracle

Impact	Short Description	Contract
Medium	LPOracle._setLPs(address[],bool[]) (contracts/Oracle/LPOracle.sol#105-116) ignores return value by IUniswapV2Pair(tokenAddresses[i]).token0() (contracts/Oracle/LPOracle.sol#111)	LPOracle
Medium	Comptroller.transferVerify(address,address, address,uint256) (contracts/Comptroller.sol#814-830) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#827)	RewardDistributor
Medium	Comptroller.repayBorrowVerify(address,address, address,uint256,uint256) (contracts/Comptroller.sol#586-604) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#601)	RewardDistributor
Medium	Comptroller.liquidateBorrowVerify(address, address,address,address,uint256,uint256) (contracts/Comptroller.sol#674-694) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#691)	RewardDistributor
Medium	Comptroller.borrowVerify(address,address, uint256) (contracts/Comptroller.sol#528-542) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#539)	RewardDistributor
Medium	Comptroller.mintVerify(address,address,uint256, uint256) (contracts/Comptroller.sol#348-364) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#361)	RewardDistributor
Medium	Comptroller.seizeVerify(address,address,address, address,uint256) (contracts/Comptroller.sol#754-772) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#769)	RewardDistributor

Impact	Short Description	Contract
Medium	RewardDistributor.setRewardSpeedInternal(uint8, CToken,uint256,uint256) (contracts/RewardDistributor.sol#153-229) uses a dangerous strict equality: - rewardSupplyState[rewardType][address(cToken)].index == 0 && rewardSupplyState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol#174-175)	RewardDistributor
Medium	RewardDistributor.setRewardSpeedInternal(uint8, CToken,uint256,uint256) (contracts/RewardDistributor.sol#153-229) uses a dangerous strict equality: - rewardBorrowState[rewardType][address(cToken)].index == 0 && rewardBorrowState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol#210-211)	RewardDistributor
Medium	Contract locking ether found: Contract RewardDistributor (contracts/RewardDistributor.sol#60-662) has payable functions: - RewardDistributor.claimReward(uint8,address[],CToken[],bool, bool) (contracts/RewardDistributor.sol#504-555) - RewardDistributor.fallback() (contracts/RewardDistributor.sol#657) But does not have a function to withdraw the ether	RewardDistributor
Medium	Comptroller.borrowAllowed(address,address, uint256).err_scope_0 (contracts/Comptroller.sol#494) is a local variable never initialized	RewardDistributor
Medium	Comptroller._supportMarket(CToken) (contracts/Comptroller.sol#1302-1334) ignores return value by cToken.isCToken() (contracts/Comptroller.sol#1319)	RewardDistributor

Impact	Short Description	Contract
Medium	Reentrancy in LockdropVaultV2.claim() (contracts/LockdropVaultV2.sol#67-76): External calls: - transferStatus = ct.transfer(msg.sender, accountBalances[msg.sender]) (contracts/LockdropVaultV2.sol#72) State variables written after the call(s): - accountBalances[msg.sender] = 0 (contracts/LockdropVaultV2.sol#74)	LockdropVaultV2
Medium	ComptrollerG1.borrowVerify(address,address, uint256) (contracts/ComptrollerG1.sol#520-534) uses a Boolean constant improperly: -false (contracts/ComptrollerG1.sol#531)	ComptrollerG1
Medium	ComptrollerG1.seizeVerify(address,address, address,address,uint256) (contracts/ComptrollerG1.sol#746-764) uses a Boolean constant improperly: -false (contracts/ComptrollerG1.sol#761)	ComptrollerG1
Medium	ComptrollerG1.mintVerify(address,address, uint256,uint256) (contracts/ComptrollerG1.sol#340-356) uses a Boolean constant improperly: -false (contracts/ComptrollerG1.sol#353)	ComptrollerG1
Medium	Comptroller.transferVerify(address,address, address,uint256) (contracts/Comptroller.sol#814-830) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#827)	ComptrollerG1
Medium	Comptroller.repayBorrowVerify(address,address, address,uint256,uint256) (contracts/Comptroller.sol#586-604) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#601)	ComptrollerG1
Medium	Comptroller.liquidateBorrowVerify(address, address,address,address,uint256,uint256) (contracts/Comptroller.sol#674-694) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#691)	ComptrollerG1

Impact	Short Description	Contract
Medium	Comptroller.borrowVerify(address,address, uint256) (contracts/Comptroller.sol#528-542) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#539)	ComptrollerG1
Medium	Comptroller.mintVerify(address,address,uint256, uint256) (contracts/Comptroller.sol#348-364) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#361)	ComptrollerG1
Medium	ComptrollerG1.repayBorrowVerify(address,address, address,uint256,uint256) (contracts/ComptrollerG1.sol#578-596) uses a Boolean constant improperly: -false (contracts/ComptrollerG1.sol#593)	ComptrollerG1
Medium	ComptrollerG1.liquidateBorrowVerify(address, address,address,address,uint256,uint256) (contracts/ComptrollerG1.sol#666-686) uses a Boolean constant improperly: -false (contracts/ComptrollerG1.sol#683)	ComptrollerG1
Medium	ComptrollerG1.transferVerify(address,address, address,uint256) (contracts/ComptrollerG1.sol #806-822) uses a Boolean constant improperly: -false (contracts/ComptrollerG1.sol#819)	ComptrollerG1
Medium	Comptroller.seizeVerify(address,address,address, address,uint256) (contracts/Comptroller.sol#754-772) uses a Boolean constant improperly: -false (contracts/Comptroller.sol#769)	ComptrollerG1
Medium	RewardDistributor.setRewardSpeedInternal(uint8, CToken,uint256,uint256) (contracts/RewardDistributor.sol#153-229) uses a dangerous strict equality: - rewardSupplyState[rewardType][address(cToken)].index == 0 && rewardSupplyState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol#174-175)	ComptrollerG1

Impact	Short Description	Contract
Medium	RewardDistributor.setRewardSpeedInternal(uint8, CToken,uint256,uint256) (contracts/RewardDistributor.sol#153-229) uses a dangerous strict equality: - rewardBorrowState[rewardType][address(cToken)].index == 0 && rewardBorrowState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol#210-211)	ComptrollerG1
Medium	Contract locking ether found: Contract RewardDistributor (contracts/RewardDistributor.sol#60-662) has payable functions: - RewardDistributor.claimReward(uint8,address[],CToken[],bool, bool) (contracts/RewardDistributor.sol#504-555) - RewardDistributor.fallback() (contracts/RewardDistributor.sol#657) But does not have a function to withdraw the ether	ComptrollerG1
Medium	ComptrollerG1.borrowAllowed(address,address, uint256).err_scope_0 (contracts/ComptrollerG1.sol#486) is a local variable never initialized	ComptrollerG1
Medium	Comptroller.borrowAllowed(address,address, uint256).err_scope_0 (contracts/Comptroller.sol#494) is a local variable never initialized	ComptrollerG1
Medium	ComptrollerG1._supportMarket(CToken) (contracts/ComptrollerG1.sol#1275-1307) ignores return value by cToken.isCToken() (contracts/ComptrollerG1.sol#1292)	ComptrollerG1
Medium	Comptroller._supportMarket(CToken) (contracts/Comptroller.sol#1302-1334) ignores return value by cToken.isCToken() (contracts/Comptroller.sol#1319)	ComptrollerG1

Impact	Short Description	Contract
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp).rewardAccrued (contracts/RewardDistributor.sol #300) shadows: - RewardDistributorStorage.rewardAccrued (contracts/RewardDistributor.sol #51) (state variable)	Comptroller
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address).rewardAccrued (contracts/RewardDistributor.sol #251) shadows: - RewardDistributorStorage.rewardAccrued (contracts/RewardDistributor.sol #51) (state variable)	Comptroller
Low	Comptroller._setBorrowCapGuardian(address).newBorrowCapGuardian (contracts/Comptroller.sol #1404) lacks a zero-check on : - borrowCapGuardian = newBorrowCapGuardian (contracts/Comptroller.sol #1411)	Comptroller
Low	Comptroller._setPauseGuardian(address).newPauseGuardian (contracts/Comptroller.sol #1422) lacks a zero-check on : - pauseGuardian = newPauseGuardian (contracts/Comptroller.sol #1438)	Comptroller
Low	Comptroller._setRewardDistributor(address).newRewardDistributor (contracts/Comptroller.sol #1104) lacks a zero-check on : - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) - rewardDistributor = newRewardDistributor (contracts/Comptroller.sol #1119)	Comptroller

Impact	Short Description	Contract
Low	RewardDistributor.setAdmin(address)._newAdmin (contracts/RewardDistributor.sol #649) lacks a zero-check on : - admin = _newAdmin (contracts/RewardDistributor.sol #651)	Comptroller
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address) (contracts/RewardDistributor.sol #236-272) has external calls inside a loop: supplyTokens = CToken(cToken).totalSupply() (contracts/RewardDistributor.sol #250)	Comptroller
Low	RewardDistributor.grantRewardInternal(uint8, address, uint256) (contracts/RewardDistributor.sol #565-579) has external calls inside a loop: rewardRemaining = reward.balanceOf(address(this)) (contracts/RewardDistributor.sol #572)	Comptroller
Low	RewardDistributor.claimReward(uint8, address[], CToken[], bool, bool) (contracts/RewardDistributor.sol #504-555) has external calls inside a loop: borrowIndex = Exp(cToken.borrowIndex()) (contracts/RewardDistributor.sol #519)	Comptroller
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #280-321) has external calls inside a loop: borrowAmount = div_-(CToken(cToken).totalBorrows(),marketBorrowIndex) (contracts/RewardDistributor.sol #296-299)	Comptroller

Impact	Short Description	Contract
Low	RewardDistributor.distributeBorrowerReward(uint8, address, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #373-415) has external calls inside a loop: borrowerAmount = div_-(CToken(cToken).borrowBalanceStored(borrower),marketBorrowIndex) (contracts/RewardDistributor.sol #398-401)	Comptroller
Low	RewardDistributor.grantRewardInternal(uint8, address, uint256) (contracts/RewardDistributor.sol #565-579) has external calls inside a loop: reward.transfer(user, amount) (contracts/RewardDistributor.sol #574)	Comptroller
Low	RewardDistributor.claimReward(uint8, address[], CToken[], bool, bool) (contracts/RewardDistributor.sol #504-555) has external calls inside a loop: require(bool, string)(comptroller.isMarketListed(address(cToken)),market must be listed) (contracts/RewardDistributor.sol #514-517)	Comptroller
Low	RewardDistributor.distributeSupplierReward(uint8, address, address) (contracts/RewardDistributor.sol #329-364) has external calls inside a loop: supplierTokens = CToken(cToken).balanceOf(supplier) (contracts/RewardDistributor.sol #350)	Comptroller
Low	Variable 'Comptroller.borrowAllowed(address, address, uint256).err (contracts/Comptroller.sol #472)' in Comptroller.borrowAllowed(address, address, uint256) (contracts/Comptroller.sol #455-520) potentially used before declaration: (err, shortfall) = getHypotheticalAccountLiquidityInternal(borrower, CToken(cToken),0,borrowAmount) (contracts/Comptroller.sol #493-502)	Comptroller

Impact	Short Description	Contract
Low	Reentrancy in Comptroller._setRewardDistributor(address) (contracts/Comptroller.sol #1104-1124): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) State variables written after the call(s): - rewardDistributor = newRewardDistributor (contracts/Comptroller.sol #1119)	Comptroller
Low	Reentrancy in RewardDistributor._grantReward(uint8, address, uint256) (contracts/RewardDistributor.sol #590-599): External calls: - amountLeft = grantRewardInternal(rewardType, recipient, amount) (contracts/RewardDistributor.sol #596) - reward.transfer(user, amount) (contracts/RewardDistributor.sol #574) Event emitted after the call(s): - RewardGranted(rewardType, recipient, amount) (contracts/RewardDistributor.sol #598)	Comptroller
Low	Reentrancy in Comptroller._setRewardDistributor(address) (contracts/Comptroller.sol #1104-1124): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) Event emitted after the call(s): - NewRewardDistributor(oldRewardDistributor, rewardDistributor) (contracts/Comptroller.sol #1121)	Comptroller
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address) (contracts/RewardDistributor.sol #236-272) uses timestamp for comparisons Dangerous comparisons: - deltaTimestamps > 0 && supplySpeed > 0 (contracts/RewardDistributor.sol #249) - deltaTimestamps > 0 (contracts/RewardDistributor.sol #266)	Comptroller

Impact	Short Description	Contract
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #280-321) uses timestamp for comparisons Dangerous comparisons: - deltaTimestamps > 0 && borrowSpeed > 0 (contracts/RewardDistributor.sol #295) - deltaTimestamps > 0 (contracts/RewardDistributor.sol #315) - borrowAmount > 0 (contracts/RewardDistributor.sol #301-303)	Comptroller
Low	FluxOracleV1.setAdmin(address).newAdmin (contracts/Oracle/FluxOracleV1.sol #100) lacks a zero-check on : - admin = newAdmin (contracts/Oracle/FluxOracleV1.sol #102)	FluxOracleV1
Low	StNearFeedV1.setStNearPrice(uint256) (contracts/Oracle/StNearFeedV1.sol #33-37) should emit an event for: - stNearPrice = _stNearPrice (contracts/Oracle/StNearFeedV1.sol #36)	StNearFeedV1
Low	StNearFeedV1.setAdmin(address).newAdmin (contracts/Oracle/StNearFeedV1.sol #39) lacks a zero-check on : - admin = newAdmin (contracts/Oracle/StNearFeedV1.sol #41)	StNearFeedV1
Low	TwapFeed.constructor(address, bool)._uniswapV2Pair (contracts/Oracle/TwapFeed.sol #39) lacks a zero-check on : - uniswapV2Pair = _uniswapV2Pair (contracts/Oracle/TwapFeed.sol #40)	TwapFeed

Impact	Short Description	Contract
Low	TwapFeed.update() (contracts/Oracle/TwapFeed.sol #115-161) uses timestamp for comparisons Dangerous comparisons: - require(bool, string)(pair.initialized, TwapFeed: NOT_INITIALIZED) (contracts/Oracle/TwapFeed.sol #119) - blockTimestamp - lastUpdateTimestamp >= MIN_T (contracts/Oracle/TwapFeed.sol #128) - require(bool, string)(T >= MIN_T, TwapFeed: NOT_READY) (contracts/Oracle/TwapFeed.sol #158)	TwapFeed
Low	TwapFeed.toUint224(uint256) (contracts/Oracle/TwapFeed.sol #93-96) uses timestamp for comparisons Dangerous comparisons: - require(bool, string)(input <= uint224(-1),TwapFeed: UINT224_OVERFLOW) (contracts/Oracle/TwapFeed.sol #94)	TwapFeed
Low	TwapFeed.constructor(address, bool) (contracts/Oracle/TwapFeed.sol #39-68) uses timestamp for comparisons Dangerous comparisons: - require(bool, string)(!pair.initialized, TwapFeed: ALREADY_INITIALIZED) (contracts/Oracle/TwapFeed.sol #47-50)	TwapFeed
Low	LPOracle._setLPs(address[], bool[]) (contracts/Oracle/LPOracle.sol #105-116) has external calls inside a loop: IUniswapV2Pair(tokenAddresses[i]).token0() (contracts/Oracle/LPOracle.sol #111)	LPOracle
Low	LPOracle._setLPs(address[], bool[]) (contracts/Oracle/LPOracle.sol #105-116) has external calls inside a loop: IUniswapV2Pair(tokenAddresses[i]).token1() (contracts/Oracle/LPOracle.sol #112)	LPOracle

Impact	Short Description	Contract
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp).rewardAccrued (contracts/RewardDistributor.sol #300) shadows: - RewardDistributorStorage.rewardAccrued (contracts/RewardDistributor.sol #51) (state variable)	RewardDistributor
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address).rewardAccrued (contracts/RewardDistributor.sol #251) shadows: - RewardDistributorStorage.rewardAccrued (contracts/RewardDistributor.sol #51) (state variable)	RewardDistributor
Low	Comptroller._setBorrowCapGuardian(address).newBorrowCapGuardian (contracts/Comptroller.sol #1404) lacks a zero-check on : - borrowCapGuardian = newBorrowCapGuardian (contracts/Comptroller.sol #1411)	RewardDistributor
Low	Comptroller._setPauseGuardian(address).newPauseGuardian (contracts/Comptroller.sol #1422) lacks a zero-check on : - pauseGuardian = newPauseGuardian (contracts/Comptroller.sol #1438)	RewardDistributor
Low	Comptroller._setRewardDistributor(address).newRewardDistributor (contracts/Comptroller.sol #1104) lacks a zero-check on : - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) - rewardDistributor = newRewardDistributor (contracts/Comptroller.sol #1119)	RewardDistributor

Impact	Short Description	Contract
Low	RewardDistributor.setAdmin(address)._newAdmin (contracts/RewardDistributor.sol #649) lacks a zero-check on : - admin = _newAdmin (contracts/RewardDistributor.sol #651)	RewardDistributor
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address) (contracts/RewardDistributor.sol #236-272) has external calls inside a loop: supplyTokens = CToken(cToken).totalSupply() (contracts/RewardDistributor.sol #250)	RewardDistributor
Low	RewardDistributor.grantRewardInternal(uint8, address, uint256) (contracts/RewardDistributor.sol #565-579) has external calls inside a loop: rewardRemaining = reward.balanceOf(address(this)) (contracts/RewardDistributor.sol #572)	RewardDistributor
Low	RewardDistributor.claimReward(uint8, address[], CToken[], bool, bool) (contracts/RewardDistributor.sol #504-555) has external calls inside a loop: borrowIndex = Exp(cToken.borrowIndex()) (contracts/RewardDistributor.sol #519)	RewardDistributor
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #280-321) has external calls inside a loop: borrowAmount = div_-(CToken(cToken).totalBorrows(),marketBorrowIndex) (contracts/RewardDistributor.sol #296-299)	RewardDistributor

Impact	Short Description	Contract
Low	RewardDistributor.distributeBorrowerReward(uint8, address, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #373-415) has external calls inside a loop: borrowerAmount = div_-(CToken(cToken).borrowBalanceStored(borrower),marketBorrowIndex) (contracts/RewardDistributor.sol #398-401)	RewardDistributor
Low	RewardDistributor.grantRewardInternal(uint8, address, uint256) (contracts/RewardDistributor.sol #565-579) has external calls inside a loop: reward.transfer(user, amount) (contracts/RewardDistributor.sol #574)	RewardDistributor
Low	RewardDistributor.claimReward(uint8, address[], CToken[], bool, bool) (contracts/RewardDistributor.sol #504-555) has external calls inside a loop: require(bool, string)(comptroller.isMarketListed(address(cToken)),market must be listed) (contracts/RewardDistributor.sol #514-517)	RewardDistributor
Low	RewardDistributor.distributeSupplierReward(uint8, address, address) (contracts/RewardDistributor.sol #329-364) has external calls inside a loop: supplierTokens = CToken(cToken).balanceOf(supplier) (contracts/RewardDistributor.sol #350)	RewardDistributor
Low	Variable 'Comptroller.borrowAllowed(address, address, uint256).err (contracts/Comptroller.sol #472)' in Comptroller.borrowAllowed(address, address, uint256) (contracts/Comptroller.sol #455-520) potentially used before declaration: (err, shortfall) = getHypotheticalAccountLiquidityInternal(borrower, CToken(cToken),0,borrowAmount) (contracts/Comptroller.sol #493-502)	RewardDistributor

Impact	Short Description	Contract
Low	Reentrancy in Comptroller._setRewardDistributor(address) (contracts/Comptroller.sol #1104-1124): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) State variables written after the call(s): - rewardDistributor = newRewardDistributor (contracts/Comptroller.sol #1119)	RewardDistributor
Low	Reentrancy in RewardDistributor._grantReward(uint8, address, uint256) (contracts/RewardDistributor.sol #590-599): External calls: - amountLeft = grantRewardInternal(rewardType, recipient, amount) (contracts/RewardDistributor.sol #596) - reward.transfer(user, amount) (contracts/RewardDistributor.sol #574) Event emitted after the call(s): - RewardGranted(rewardType, recipient, amount) (contracts/RewardDistributor.sol #598)	RewardDistributor
Low	Reentrancy in Comptroller._setRewardDistributor(address) (contracts/Comptroller.sol #1104-1124): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) Event emitted after the call(s): - NewRewardDistributor(oldRewardDistributor, rewardDistributor) (contracts/Comptroller.sol #1121)	RewardDistributor
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address) (contracts/RewardDistributor.sol #236-272) uses timestamp for comparisons Dangerous comparisons: - deltaTimestamps > 0 && supplySpeed > 0 (contracts/RewardDistributor.sol #249) - deltaTimestamps > 0 (contracts/RewardDistributor.sol #266)	RewardDistributor

Impact	Short Description	Contract
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #280-321) uses timestamp for comparisons Dangerous comparisons: - deltaTimestamps > 0 && borrowSpeed > 0 (contracts/RewardDistributor.sol #295) - deltaTimestamps > 0 (contracts/RewardDistributor.sol #315) - borrowAmount > 0 (contracts/RewardDistributor.sol #301-303)	RewardDistributor
Low	NEAROracle.setAdmin(address).newAdmin (contracts/Oracle/NEAROracle.sol #112) lacks a zero-check on : - admin = newAdmin (contracts/Oracle/NEAROracle.sol #114)	NEAROracle
Low	FluxOracle.setAdmin(address).newAdmin (contracts/Oracle/FluxOracle.sol #110) lacks a zero-check on : - admin = newAdmin (contracts/Oracle/FluxOracle.sol #112)	FluxOracle
Low	LockdropVaultV2.setOwner(address) (contracts/LockdropVaultV2.sol #82-86) should emit an event for: - owner = newOwner (contracts/LockdropVaultV2.sol #85)	LockdropVaultV2
Low	LockdropVaultV2.constructor(string, address, uint256).ctoken_ (contracts/LockdropVaultV2.sol #29) lacks a zero-check on : - ctoken = ctoken_ (contracts/LockdropVaultV2.sol #34)	LockdropVaultV2

Impact	Short Description	Contract
Low	Reentrancy in LockdropVaultV2.deposit(uint256) (contracts/LockdropVaultV2.sol #48-58): External calls: - transferStatus = ct.transferFrom(msg.sender, address(this),amount) (contracts/LockdropVaultV2.sol #52) State variables written after the call(s): - accountBalances[msg.sender] = accountBalances[msg.sender].add(amount) (contracts/LockdropVaultV2.sol #54) - totalDeposits = totalDeposits.add(amount) (contracts/LockdropVaultV2.sol #55)	LockdropVaultV2
Low	Reentrancy in LockdropVaultV2.deposit(uint256) (contracts/LockdropVaultV2.sol #48-58): External calls: - transferStatus = ct.transferFrom(msg.sender, address(this),amount) (contracts/LockdropVaultV2.sol #52) Event emitted after the call(s): - Deposit(msg.sender, amount) (contracts/LockdropVaultV2.sol #56)	LockdropVaultV2
Low	Reentrancy in LockdropVaultV2.claim() (contracts/LockdropVaultV2.sol #67-76): External calls: - transferStatus = ct.transfer(msg.sender, accountBalances[msg.sender]) (contracts/LockdropVaultV2.sol #72) Event emitted after the call(s): - Claim(msg.sender, claimAnnouncement) (contracts/LockdropVaultV2.sol #75)	LockdropVaultV2
Low	LockdropVaultV2.constructor(string, address, uint256) (contracts/LockdropVaultV2.sol #28-37) uses timestamp for comparisons Dangerous comparisons: - require(bool, string)(claimUnlockTime_ > now, claim unlock time is before current time) (contracts/LockdropVaultV2.sol #32)	LockdropVaultV2

Impact	Short Description	Contract
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp).rewardAccrued (contracts/RewardDistributor.sol #300) shadows: - RewardDistributorStorage.rewardAccrued (contracts/RewardDistributor.sol #51) (state variable)	ComptrollerG1
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address).rewardAccrued (contracts/RewardDistributor.sol #251) shadows: - RewardDistributorStorage.rewardAccrued (contracts/RewardDistributor.sol #51) (state variable)	ComptrollerG1
Low	Comptroller._setBorrowCapGuardian(address).newBorrowCapGuardian (contracts/Comptroller.sol #1404) lacks a zero-check on : - borrowCapGuardian = newBorrowCapGuardian (contracts/Comptroller.sol #1411)	Comptroller
Low	ComptrollerG1._setPauseGuardian(address).newPauseGuardian (contracts/ComptrollerG1.sol #1395) lacks a zero-check on : - pauseGuardian = newPauseGuardian (contracts/ComptrollerG1.sol #1411)	ComptrollerG1
Low	Comptroller._setPauseGuardian(address).newPauseGuardian (contracts/Comptroller.sol #1422) lacks a zero-check on : - pauseGuardian = newPauseGuardian (contracts/Comptroller.sol #1438)	ComptrollerG1

Impact	Short Description	Contract
Low	ComptrollerG1._setRewardDistributor(address).newRewardDistributor (contracts/ComptrollerG1.sol #1096) lacks a zero-check on : - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/ComptrollerG1.sol #1103-1105) - rewardDistributor = newRewardDistributor (contracts/ComptrollerG1.sol #1111)	ComptrollerG1
Low	Comptroller._setRewardDistributor(address).newRewardDistributor (contracts/Comptroller.sol #1104) lacks a zero-check on : - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) - rewardDistributor = newRewardDistributor (contracts/Comptroller.sol #1119)	Comptroller
Low	ComptrollerG1._setBorrowCapGuardian(address).newBorrowCapGuardian (contracts/ComptrollerG1.sol #1377) lacks a zero-check on : - borrowCapGuardian = newBorrowCapGuardian (contracts/ComptrollerG1.sol #1384)	ComptrollerG1
Low	RewardDistributor.setAdmin(address)._newAdmin (contracts/RewardDistributor.sol #649) lacks a zero-check on : - admin = _newAdmin (contracts/RewardDistributor.sol #651)	ComptrollerG1
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address) (contracts/RewardDistributor.sol #236-272) has external calls inside a loop: supplyTokens = CToken(cToken).totalSupply() (contracts/RewardDistributor.sol #250)	ComptrollerG1

Impact	Short Description	Contract
Low	RewardDistributor.grantRewardInternal(uint8, address, uint256) (contracts/RewardDistributor.sol #565-579) has external calls inside a loop: rewardRemaining = reward.balanceOf(address(this)) (contracts/RewardDistributor.sol #572)	ComptrollerG1
Low	RewardDistributor.claimReward(uint8, address[], CToken[], bool, bool) (contracts/RewardDistributor.sol #504-555) has external calls inside a loop: borrowIndex = Exp(cToken.borrowIndex()) (contracts/RewardDistributor.sol #519)	ComptrollerG1
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #280-321) has external calls inside a loop: borrowAmount = div_-(CToken(cToken).totalBorrows(),marketBorrowIndex) (contracts/RewardDistributor.sol #296-299)	ComptrollerG1
Low	RewardDistributor.distributeBorrowerReward(uint8, address, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #373-415) has external calls inside a loop: borrowerAmount = div_-(CToken(cToken).borrowBalanceStored(borrower),marketBorrowIndex) (contracts/RewardDistributor.sol #398-401)	ComptrollerG1
Low	RewardDistributor.grantRewardInternal(uint8, address, uint256) (contracts/RewardDistributor.sol #565-579) has external calls inside a loop: reward.transfer(user, amount) (contracts/RewardDistributor.sol #574)	ComptrollerG1

Impact	Short Description	Contract
Low	RewardDistributor.claimReward(uint8, address[], CToken[], bool, bool) (contracts/RewardDistributor.sol #504-555) has external calls inside a loop: require(bool, string)(comptroller.isMarketListed(address(cToken)),market must be listed) (contracts/RewardDistributor.sol #514-517)	ComptrollerG1
Low	RewardDistributor.distributeSupplierReward(uint8, address, address) (contracts/RewardDistributor.sol #329-364) has external calls inside a loop: supplierTokens = CToken(cToken).balanceOf(supplier) (contracts/RewardDistributor.sol #350)	ComptrollerG1
Low	Variable 'Comptroller.borrowAllowed(address, address, uint256).err (contracts/Comptroller.sol #472)' in Comptroller.borrowAllowed(address, address, uint256) (contracts/Comptroller.sol #455-520) potentially used before declaration: (err, shortfall) = getHypotheticalAccountLiquidityInternal(borrower, CToken(cToken),0,borrowAmount) (contracts/Comptroller.sol #493-502)	ComptrollerG1
Low	Variable 'ComptrollerG1.borrowAllowed(address, address, uint256).err (contracts/ComptrollerG1.sol #464)' in ComptrollerG1.borrowAllowed(address, address, uint256) (contracts/ComptrollerG1.sol #447-512) potentially used before declaration: (err, shortfall) = getHypotheticalAccountLiquidityInternal(borrower, CToken(cToken),0,borrowAmount) (contracts/ComptrollerG1.sol #485-494)	ComptrollerG1

Impact	Short Description	Contract
Low	Reentrancy in Comptroller._setRewardDistributor(address) (contracts/Comptroller.sol #1104-1124): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) State variables written after the call(s): - rewardDistributor = newRewardDistributor (contracts/Comptroller.sol #1119)	ComptrollerG1
Low	Reentrancy in RewardDistributor._grantReward(uint8, address, uint256) (contracts/RewardDistributor.sol #590-599): External calls: - amountLeft = grantRewardInternal(rewardType, recipient, amount) (contracts/RewardDistributor.sol #596) - reward.transfer(user, amount) (contracts/RewardDistributor.sol #574) Event emitted after the call(s): - RewardGranted(rewardType, recipient, amount) (contracts/RewardDistributor.sol #598)	ComptrollerG1
Low	Reentrancy in Comptroller._setRewardDistributor(address) (contracts/Comptroller.sol #1104-1124): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/Comptroller.sol #1111-1113) Event emitted after the call(s): - NewRewardDistributor(oldRewardDistributor, rewardDistributor) (contracts/Comptroller.sol #1121)	ComptrollerG1
Low	RewardDistributor.updateRewardSupplyIndex(uint8, address) (contracts/RewardDistributor.sol #236-272) uses timestamp for comparisons Dangerous comparisons: - deltaTimestamps > 0 && supplySpeed > 0 (contracts/RewardDistributor.sol #249) - deltaTimestamps > 0 (contracts/RewardDistributor.sol #266)	ComptrollerG1

Impact	Short Description	Contract
Low	RewardDistributor.setRewardSpeedInternal(uint8, CToken, uint256, uint256) (contracts/RewardDistributor.sol #153-229) uses timestamp for comparisons Dangerous comparisons: - rewardSupplyState[rewardType][address(cToken)].index == 0 && rewardSupplyState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol #174-175) - rewardBorrowState[rewardType][address(cToken)].index == 0 && rewardBorrowState[rewardType][address(cToken)].timestamp == 0 (contracts/RewardDistributor.sol #210-211)	ComptrollerG1
Low	RewardDistributor.updateRewardBorrowIndex(uint8, address, ExponentialNoError.Exp) (contracts/RewardDistributor.sol #280-321) uses timestamp for comparisons Dangerous comparisons: - deltaTimestamps > 0 && borrowSpeed > 0 (contracts/RewardDistributor.sol #295) - deltaTimestamps > 0 (contracts/RewardDistributor.sol #315) - borrowAmount > 0 (contracts/RewardDistributor.sol #301-303)	ComptrollerG1
Low	Reentrancy in ComptrollerG1._setRewardDistributor(address) (contracts/ComptrollerG1.sol #1096-1116): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/ComptrollerG1.sol #1103-1105) Event emitted after the call(s): - NewRewardDistributor(oldRewardDistributor, rewardDistributor) (contracts/ComptrollerG1.sol #1113)	ComptrollerG1

Impact	Short Description	Contract
Low	Reentrancy in ComptrollerG1._setRewardDistributor(address) (contracts/ComptrollerG1.sol #1096-1116): External calls: - (success) = newRewardDistributor.call.value(0)(abi.encodeWithSignature(initialize(),0)) (contracts/ComptrollerG1.sol #1103-1105) State variables written after the call(s): - rewardDistributor = newRewardDistributor (contracts/ComptrollerG1.sol #1111)	ComptrollerG1
Low	LockdropVaultV2.claim() (contracts/LockdropVaultV2.sol #67-76) uses timestamp for comparisons Dangerous comparisons: - require(bool, string)(now > claimUnlockTime, Claim Functionality Still Locked) (contracts/LockdropVaultV2.sol #68)	LockdropVaultV2

THANK YOU FOR CHOOSING

// HALBORN