HALBORN

NewOrderDAO - Sector Tokens + Merkle reward distributor + Voting Escrow

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 13th, 2023 - March 23rd, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/15/2023	Francisco González
0.2	Document Creation	03/21/2023	Francisco González
0.3	Document Creation	03/22/2023	Francisco González
0.4	Draft Review	03/23/2023	Ataberk Yavuzer
0.5	Draft Review	03/23/2023	Piotr Cielas
0.6	Draft Review	03/23/2023	Gabi Urrutia
1.0	Remediation Plan	03/24/2023	Francisco González
1.1	Remediation Plan Review	03/24/2023	Ataberk Yavuzer
1.2	Remediation Plan Review	03/25/2023	Piotr Cielas
1.3	Remediation Plan Review	03/27/2023	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
Francisco González	Halborn	Francisco.Villarejo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

NewOrderDAO is a community-led incubation DAO that builds edge-of-the-edge Web3 projects. Sector Finance creates risk tools and investment products to empower the next generation of DeFi users. In this engagement, ERC20 Tokens, Merkle Reward distributor and Voting Escrow mechanisms will be audited.

NewOrderDAO engaged Halborn to conduct a security audit on Sector Finance smart contracts beginning on March 13th, 2023 and ending on March 23rd, 2023 . The security assessment was scoped to the smart contracts and functions detailed in the Scope section of this report, along with Commit hashes and further details.

1.2 AUDIT SUMMARY

The team at Halborn was provided 2 weeks for the engagement and assigned a full-time security engineer to audit the security of the programs in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audits is to:

- Identify potential security issues within the programs
- Ensure that smart contract functions operate as intended

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Sector Finance team. The main ones are the following:

- Enforcing that blocklist address has been set on VotingEscrow contract.
- Ensuring that a merkleRoot has been set before executing claim() function.

- Adjust rounding logic.

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 this audit. 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 the security best practices. The following phases and associated tools were used during 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 hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE, Ganache)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. 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

Code repositories:

1. Repository: sector-fi/sector-token

- Commit ID: c03f66e4c76540de07b8ee853c77851a916d1acb
- Smart contracts in scope:
  1. bSECT.sol
  2. lveSECT.sol
  3. RewardDistributor.sol
  4. SECT.sol
  5. VotingEscrow.sol
     Scoped contract changes:
     - Added lveSECToken storage for a contract that can call lockFor on behalf of a user.
     - Added updateLveSECT method - only owner can update lveSECToken .
     - Added lockFor that allows a smart contract to lock tokens on behalf of a user.
     - _lockFor logic requirement relaxed to allow update non-empty, non-delegated locks.
     - Added increaseAmountFor method that anyone can call to add more tokens to a lock.

2. Updated Commit ID: 8a98dc920bd5e7838bdcd9ac6cbf8c1ad6d3f197
   Scoped contract changes:
   - Duration can be defined lveSECT instead of being fixed to ~6 months.
   - Instead of limiting the usage of lockFor() to a single lveSECT address, a mapping containing whitelisted addresses is now used.

3. Updated Commit ID 2: bf1f0de01855fbca1431f8750c2e35f49ea17fe1
   Scoped contract changes:
   - SECT has been simplified, minting full supply to contract deployer.

4. Remediations Commit ID: 7c13bd7233895a29d94f781af3ceba01742aeca1

Out-of-scope:

- Third-party libraries and dependencies
- Economical attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	5	11

LIKELIHOOD

IMPACT

(HAL-02)				
(HAL-03) (HAL-04) (HAL-05) (HAL-06)		(HAL-01)		
(HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17)	(HAL-07)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) - INCONSISTENT BEHAVIOR IF NO BLOCKLIST IS DEFINED IN THE VOTINGESCROW CONTRACT	Medium	SOLVED - 03/24/2023
(HAL-02) - LACK OF PARAMETER LIMITS	Low	PARTIALLY SOLVED - 03/24/2023
(HAL-03) - ROUNDING CAN LEAD TO TRANSACTION REVERTS	Low	RISK ACCEPTED
(HAL-04) - LACK OF TWO STEP OWNERSHIP TRANSFER PATTERN	Low	RISK ACCEPTED
(HAL-05) - OWNER CAN RENOUNCE OWNERSHIP	Low	RISK ACCEPTED
(HAL-06) - FLOATING PRAGMA	Low	SOLVED - 03/24/2023
(HAL-07) - USERS CALLING THE QUITLOCK FUNCTION MAY NOT BE ABLE TO CONVERT LVESECT TOKENS	Informational	SOLVED - 03/24/2023
(HAL-08) - THE CLAIM() FUNCTION CAN BE CALLED WHEN MERKLE ROOT IS ZERO	Informational	SOLVED - 03/24/2023
(HAL-09) - REDUNDANT FUNCTION PARAMETER	Informational	SOLVED - 03/24/2023
(HAL-10) - OPEN TO-DO	Informational	SOLVED - 03/24/2023
(HAL-11) - REDUNDANT INITIALIZATION OF UINT AND INT VARIABLES TO 0	Informational	ACKNOWLEDGED
(HAL-12) - CONTRACT PAUSE FEATURE MISSING	Informational	ACKNOWLEDGED
(HAL-13) - UNUSED LIBRARIES	Informational	SOLVED - 03/24/2023
(HAL-14) - FOR LOOPS GAS OPTIMIZATION	Informational	ACKNOWLEDGED
(HAL-15) - INCOMPLETE NATSPEC DOCUMENTATION	Informational	ACKNOWLEDGED
(HAL-16) - IMMUTABLE KEYWORD CAN BE INTRODUCED	Informational	SOLVED - 03/24/2023

(HAL-17) - ANYONE CAN CLAIM REWARDS FOR ANY USER	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) INCONSISTENT BEHAVIOR IF NO BLOCKLIST IS DEFINED IN THE VOTINGESCROW CONTRACT - MEDIUM

Description:

In the VotingEscrow contract, an instance of the Blocklist contract can be used to blocklist specific contract addresses, preventing them from interacting with the VotingEscrow.sol contract.

It has been detected that, since the address of the Blocklist contract is not initialized when deploying the contract (and can also be set to the zero address at any time), VotingEscrow can operate with the zero address as the blocklist address.
This is partially foreseen and handled in checkBlocklist() modifier:

Listing 1: VotingEscrow.sol (Line 138)

```
137     modifier checkBlocklist() {
138         if (blocklist != address(0))
139             require(!IBlocklist(blocklist).isBlocked(msg.sender),
 ↳ "Blocked contract");
140         _;
141     }
```

This would allow any function that uses the checkBlocklist() modifier to work properly when the blocklist address is set to zero. However, while functions such as createLock() or lockFor() execute without any problem, the delegate() function always reverts, as this function makes an additional unhandled call to blocklist.

This would put the contract in an inconsistent state, where locks could be created or deleted, but not delegated.

Code Location:

Listing 2: VotingEscrow.sol (Line 610)

```
598     /// @notice Delegate lock and voting power to another lock
599     /// The receiving lock needs to have a longer lock duration
600     /// The delegated lock will inherit the receiving lock's
 ↳ expiration
601     /// @param _addr The address of the lock owner to which to
 ↳ delegate
602     function delegate(address _addr) external override
 ↳ nonReentrant checkBlocklist {
603         // Different restrictions apply to undelegation
604         if (_addr == msg.sender) {
605             _undelegate();
606             return;
607         }
608         LockedBalance memory locked_ = locked[msg.sender];
609         // Validate inputs
610         require(!IBlocklist(blocklist).isBlocked(_addr), "Blocked
 ↳ contract");
611         require(locked_.amount > 0, "No lock");
612         require(locked_.end > block.timestamp, "Lock expired");
613         require(locked_.delegatee != _addr, "Already delegated");
614         // Update locks
615         int128 value = locked_.amount;
616         address delegatee = locked_.delegatee;
617         LockedBalance memory toLocked = locked[_addr];
618         locked_.delegatee = _addr;
619         if (delegatee != msg.sender) {
620             locked[msg.sender] = locked_;
621             locked_ = locked[delegatee];
622         }
623         require(toLocked.amount > 0, "Delegatee has no lock");
624         require(toLocked.end > block.timestamp, "Delegatee lock
 ↳ expired");
625         require(toLocked.end >= locked_.end, "Only delegate to
 ↳ longer lock");
626         _delegate(delegatee, locked_, value, LockAction.UNDELEGATE
 ↳ );
627         _delegate(_addr, toLocked, value, LockAction.DELEGATE);
628     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to enforce that the blocklist is a valid contract address to ensure both security and correct contract behavior.

On the other hand, if some functionality must be paused, it is recommended to implement more explicit modifiers such as whenDelegationNotPaused(), rather than relying on indirect restrictions like this.

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by allowing the VotingEscrow contract to work properly without any blocklist address set in commit bf1f0de01855fbca1431f8750c2e35f49ea17fe1.

3.2 (HAL-02) LACK OF PARAMETER LIMITS - LOW

Description:

It has been detected that some parameter-modifying functions do not implement sanity checks of the contract variables they set. This may cause the contract to function with parameter values that, although allowed, make no sense in the application context, which might cause various problems or even render the contract unusable.

Code Location:

Listing 3: bSECT.sol

```
23     /// @dev price must be set immediately upon liquidity
↳ deployment
24     function setPrice(uint256 price_) public onlyOwner {
25         price = price_;
26         emit SetPrice(price_);
27     }
```

This function's lack of sanity check allows owner to set a USDC price for converting bSECT into SECT as high or low as he wants (always greater than zero). For example, if the price was mistakenly set to 1 (instead of 1e18), users could convert bSECT into SECT virtually for free. On the other hand, if a malicious owner sets a high price just before a legitimate convert call is processed (via frontrunning), the complete USDC balance of the user could be drained (if an infinite approval is set).

Something similar happens to duration parameter in lveSECT.sol:

Listing 4: lveSECT.sol

```
16     constructor(address sect_, uint256 duration_) ERC20("liquid
↳ veSECT", "lveSECT") {
```

```
17          sect = IERC20(sect_);
18          duration = duration_;
19      }
```

In this scenario, if a duration value greater than 2 YEARS is mistakenly set, the convertToLock() function calls always revert (due to the VotingEscrow contract's MAXTIME constant). On the other hand, if lower values of duration are set, inaccuracies could be introduced due to the VotingEscrow contract's duration parameter rounding down to weeks.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to enforce logical value limits for critical parameters and check for additional occurrences of this same vulnerability.

Remediation Plan:

PARTIALLY SOLVED: The Sector Finance team partially solved this finding by restricting the value of the duration parameter to the range between 1 week and 2 years in the lveSECT contract.

However, in the bSECT contract, only one comment was added to improve code readability and avoid errors.

Commit ID 1: 98e96df2e2214de5fe709515156c741370387587

Commit ID 2: 7c13bd7233895a29d94f781af3ceba01742aeca1

3.3 (HAL-03) ROUNDING CAN LEAD TO TRANSACTION REVERTS - LOW

Description:

In the bSECT contract, the convert() function allows bSECT holders to convert their bSECT balance into SECT at a determined price (paid in USDC at the time of writing this report).

However, it has been detected that, due to how the amount due (underlyingAmnt) is calculated, the convert() function calls for low bSECT amounts (lower than ~10^12 wei) could return an underlyingAmnt of 0, allowing users to convert tiny amounts of bSECT into SECT for free. To avoid this, underlyingAmnt is increased by 1 if (amount * price)% 1e18 > 0, which prevents underlyingAmnt from being 0. However, this also increases underlyingAmnt by 1 when converting non-round amounts, which could cause reverts if an aproval for only (amount * price)/1e18 was set.

Code Location:

Listing 5: bSECT.sol (Line 40)

```
35     function convert(uint256 amount) public {
36         if (price == 0) revert PriceNotSet();
37         _burn(msg.sender, amount);
38         uint num = amount * price;
39         // round up to avoid griefing
40         uint underlyingAmnt = num / 1e18 + (num % 1e18 > 0 ? 1 :
↳ 0);
41         underlying.safeTransferFrom(msg.sender, address(this),
↳ underlyingAmnt);
42         SECT.transfer(msg.sender, amount);
43         emit Convert(msg.sender, amount);
44     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to apply the rounding up only when needed, establishing minimum prices for tiny `convert()` function calls. Otherwise, this behavior should be well documented (or even include a `calculateConvertPrice()` function) to prevent possible price miscalculations when interacting with the contract.

Remediation Plan:

RISK ACCEPTED: The Sector Finance team accepted the risk of this finding, delegating security controls to the front-end of the application.

3.4 (HAL-04) LACK OF TWO STEP OWNERSHIP TRANSFER PATTERN - LOW

Description:

When transferring the ownership of the protocol, no checks are performed on whether the new address is valid and active. In case there is a mistake when transferring the ownership, the whole protocol may lose all of its ownership functionalities.

Code Location:

Listing 6: Ownable.sol

```
69     /**
70      * @dev Transfers ownership of the contract to a new account
↳ (`newOwner`).
71      * Can only be called by the current owner.
72      */
73     function transferOwnership(address newOwner) public virtual
↳ onlyOwner {
74         require(newOwner != address(0), "Ownable: new owner is the
↳  zero address");
75         _transferOwnership(newOwner);
76     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

The transfer of ownership process should be split into two different transactions, the first one calling the requestTransferOwnership function which proposes a new owner for the protocol, and the second one, the new

owner accepts the proposal by calling acceptsTransferOwnership function. OpenZeppelin's ownable2step.sol library could be used to implement the mentioned process.

Remediation Plan:

RISK ACCEPTED: The Sector Finance team accepted the risk of this finding.

3.5 (HAL-05) OWNER CAN RENOUNCE OWNERSHIP - LOW

Description:

The owner of a contract is usually the account that deploys the contract. As a result, the owner can perform some privileged functions. In the scoped contracts, the renounceOwnership function could be used to renounce the owner permission. Renouncing ownership before transferring would result in the contract having no owner, eliminating the ability to call privileged functions.

Code Location:

Listing 7: openzeppelin-contracts/contracts/access/Ownable.sol

```
61     function renounceOwnership() public virtual onlyOwner {
62         _transferOwnership(address(0));
63     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended that the owner cannot call renounceOwnership without first transferring Ownership to another address.

Remediation Plan:

RISK ACCEPTED: The Sector Finance team accepted the risk of this finding.

3.6 (HAL-06) FLOATING PRAGMA - LOW

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Code Location:

bSECT.sol
- Line 2: `pragma solidity ^0.8.16;`

lveSECT.sol
- Line 2: `pragma solidity ^0.8.16;`

RewardDistributor.sol
- Line 2: `pragma solidity ^0.8.16;`

SECT.sol
- Line 2: `pragma solidity ^0.8.16;`

VotingEscrow.sol
- Line 2: `pragma solidity ^0.8.3;`

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider locking the pragma version in the smart contracts. It is not recommended to use a floating pragma in production. In addition, using the same pragma version for every smart contract is highly recommended.

For example: pragma solidity 0.8.19;

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by fixing the scoped contract's pragma version to 0.8.16 in Commit bf1f0de01855fbca1431f8750c2e35f49ea17fe1.

3.7 (HAL-07) USERS CALLING THE QUITLOCK FUNCTION MAY NOT BE ABLE TO CONVERT LVESECT TOKENS - INFORMATIONAL

Description:

When a user calls the quitLock() function, locked.amount is 0, and the lock expiry date is maintained (as stated in VotingEscrow.sol:694).

However, it has been noted that, if the lock expiry date is still greater than the lveSECT locking period, calls to both convertToLock() and addValueToLock() revert, rendering lveSECT non-claimable until locked.end - block.timestamp <= lveSECT.duration. This is caused by the convertToLock() reverting with the Only increase lock end message since unlock_time is lower than locked_.end, and addValueToLock() reverts with the No lock message, since locked_.amount = 0.

Code Location:

Listing 8: VotingEscrow.sol (Line 465)

```
452     function _lockFor(
453         address account,
454         uint256 _value,
455         uint256 _unlockTime
456     ) internal {
457         uint256 unlock_time = _floorToWeek(_unlockTime); //
 ↳ Locktime is rounded down to weeks
458         LockedBalance memory locked_ = locked[account];
459         LockedBalance memory oldLock = _copyLock(locked_);
460         // Validate inputs
461         require(_value != 0, "Only non zero amount");
462         // require(locked_.amount == 0, "Lock exists");
463         // we can relax the above condition for non-delegated
 ↳ accounts
464         require(locked_.delegatee == account || locked_.amount ==
```

```
 ↳ 0, "Delegated lock"); /
465         require(unlock_time >= locked_.end, "Only increase lock
 ↳ end"); // from using quitLock, user should increaseAmount instead
466         require(unlock_time > block.timestamp, "Only future lock
 ↳ end");
467         require(unlock_time <= block.timestamp + MAXTIME, "Exceeds
 ↳  maxtime");
468         // Update total supply of token deposited
469         supply = supply + _value;
470 ...
```

Listing 9: VotingEscrow.sol (Line 515)

```
511     function _increaseAmount(address account, uint256 _value)
 ↳ internal {
512         LockedBalance memory locked_ = locked[account];
513         // Validate inputs
514         require(_value != 0, "Only non zero amount");
515         require(locked_.amount > 0, "No lock");
516         require(locked_.end > block.timestamp, "Lock expired");
517         // Update total supply of token deposited
518         supply = supply + _value;
519 ...
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Although this behavior can be easily circumvented by calling VotingEscrow.createLock(1 wei, (chain.timestamp + duration), thus enabling the usage of addValueToLock() function to lock lveSECT, it is recommended to document this behavior, so users are aware of it when calling quitLock(), since mitigating it at smart contract level would require from some refactoring that may break some invariants of the contract.

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by highlighting this behavior in a comment in the lveSECT contract in commit 7c13bd7233895a29d94f781af3ceba01742aeca1.

3.8 (HAL-08) THE CLAIM() FUNCTION CAN BE CALLED WHEN MERKLE ROOT IS ZERO - INFORMATIONAL

Description:

The RewardDistributor contract uses Merkle Trees to allow users to claim earned rewards, validating amounts and receivers.

The only function contained in the contract (besides updateMerkleRoot()) is claim(), which allows any user to claim a reward if the correct address, totalAmount, and merkleProof are provided.

However, it has been detected that the claim() function can be called even when no merkleRoot was set. Although no risks have been identified because of that, adding a require statement checking that a valid merkleRoot has been set before executing the actual claim, which would save gas costs from users calling the function when no merkleRoot is set.

Code Location:

Listing 10: RewardDistributor.sol

```
42     function claim(
43         uint256 index,
44         address account,
45         uint256 totalAmount,
46         bytes32[] calldata merkleProof
47     ) external override {
48         require(claimed[account] < totalAmount, "MerkleDistributor
↳ : Nothing to claim");
49
50         // this is gearbox merkle version
51         // bytes32 node = keccak256(abi.encodePacked(account,
↳ totalAmount));
52         bytes32 node = keccak256(bytes.concat(keccak256(abi.encode
↳ (account, totalAmount))));
53
```

```
54          require(
55              MerkleProof.verify(merkleProof, merkleRoot, node),
56              "MerkleDistributor: Invalid proof."
57          );
58
59          uint256 claimedAmount = totalAmount - claimed[account];
60          claimed[account] += claimedAmount;
61
62          // TODO: wrap token into bToken and lveToken and
↳ distribute 1/2 of each
63          uint bTokenAmount = claimedAmount / 2;
64          uint lveTokenAmount = claimedAmount - bTokenAmount;
65          bToken.mintTo(account, bTokenAmount);
66          lveToken.mintTo(account, lveTokenAmount);
67
68          emit Claimed(account, claimedAmount, false);
69      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding a `require` statement first thing on `claim()` function (or a modifier) to ensure that a valid `merkleRoot` has been set. In addition, if correctly documented, that statement or modifier could be used as a pausing mechanism for `claim()` function.

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by adding a `require` statement that checks if `merkleRoot` is set before calling the `claim()` function in commit `bf1f0de01855fbca1431f8750c2e35f49ea17fe1`.

3.9 (HAL-09) REDUNDANT FUNCTION PARAMETER - INFORMATIONAL

Description:

The index function parameter is not used anywhere in the claim() in the RewardDistributor contract. Redundant function parameters decrease code readability and increase both deployment and execution gas costs.

Code Location:

Listing 11: RewardDistributor.sol (Line 43)

```
42     function claim(
43         uint256 index,
44         address account,
45         uint256 totalAmount,
46         bytes32[] calldata merkleProof
47     ) external override {
48 ...
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If not needed, it is recommended to remove unused function parameters to improve code readability and reduce gas costs.

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by removing the redundant index parameter in Commit bf1f0de01855fbca1431f8750c2e35f49ea17fe1.

3.10 (HAL-10) OPEN TO-DO - INFORMATIONAL

Description:

Open TO-DOs can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

Listing 12: RewardDistributor.sol

```
62          // TODO: wrap token into bToken and lveToken and
↳ distribute 1/2 of each
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving the TO-DOs before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by removing the TO-DO in Commit bf1f0de01855fbca1431f8750c2e35f49ea17fe1.

3.11 (HAL-11) REDUNDANT INITIALIZATION OF UINT AND INT VARIABLES TO 0 - INFORMATIONAL

Description:

Any variable of type `uint` or `int` is already initialized to `0` when declared. `uint256 i = 0` reassigns the `0` to `i` which wastes gas.

VotingEscrow.sol
- Line 248: `int128 oldSlopeDelta = 0;`
- Line 249: `int128 newSlopeDelta = 0;`
- Line 316: `uint256 blockSlope = 0; // dblock/dt`
- Line 331: `int128 dSlope = 0;`
- Line 762: `uint256 min = 0;`
- Line 784: `uint256 min = 0;`
- Line 848: `uint256 dBlock = 0;`
- Line 849: `uint256 dTime = 0;`
- Line 884: `int128 dSlope = 0;`
- Line 940: `uint256 dTime = 0;`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

To save some gas, `int` and `uint` variables can be not initialized to `0` to save some gas. For example, use instead:
`for (uint256 i; i < proposal.targets.length; ++i)`.

Remediation Plan:

ACKNOWLEDGED: The Sector Finance team acknowledged this issue.

3.12 (HAL-12) CONTRACT PAUSE FEATURE MISSING - INFORMATIONAL

Description:

It was identified that the Owner cannot pause any of the scoped contracts. In the event of a security incident, the owner would not be able to prevent the withdrawals of approved invoices. Pausing the contract can also lead to more considered decisions.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding the pausable functionality to the contract.

Remediation Plan:

ACKNOWLEDGED: The Sector Finance team acknowledged this issue.

3.13 (HAL-13) UNUSED LIBRARIES - INFORMATIONAL

Description:

Multiple unused library imports were identified in the contracts:

lveSECT.sol:
- hardhat/console.sol

SECT.sol:
- SafeERC20

Unused imports decrease the readability of the contracts.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to review the contracts and remove any unnecessary imports from them.

Remediation Plan:

SOLVED: The Sector Finance team solved this finding by removing unused libraries in commit bf1f0de01855fbca1431f8750c2e35f49ea17fe1.

3.14 (HAL-14) FOR LOOPS GAS OPTIMIZATION - INFORMATIONAL

Description:

In `for` loops, `=!` can be used instead of `<` in exit conditions to save gas.

Identified instances:

VotingEscrow.sol
- Line 327: `for (uint256 i; i < 255; ){`
- Line 765: `for (uint256 i; i < 128; ){`
- Line 786: `for (uint256 i; i < 128; ){`
- Line 882: `for (uint256 i; i < 255; ){`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use `!=` instead of `<` in the exit conditions to save gas.

Remediation Plan:

ACKNOWLEDGED: The Sector Finance team acknowledged this issue.

3.15 (HAL-15) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

NatSpec documentation is useful for internal developers that need to work on the project, external developers that need to integrate with the project, auditors that have to review it but also for end users given that many of the main blockchain explorers have officially integrated the support for it directly on their site.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding the missing NatSpec documentation.

Remediation Plan:

ACKNOWLEDGED: The Sector Finance team acknowledged this finding. The NatSpec documentation may be added in a future release.

3.16 (HAL-16) IMMUTABLE KEYWORD CAN BE INTRODUCED - INFORMATIONAL

Description:

The assessment revealed several items in code that can be declared as immutable. The compiler does not reserve a storage slot for these variables, saving gas costs.

Code Location:

Listing 13: lveSECT.sol

```
14     uint256 public duration;
```

Listing 14: VotingEscrow.sol

```
80     uint256 public decimals;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to apply immutable modifier to save some gas.

Remediation Plan:

SOLVED: The Sector Finance team solved this issue in commit 98e96df2e2214de5fe709515156c741370387587: all instances of identified parameters now use the immutable modifier.

3.17 (HAL-17) ANYONE CAN CLAIM REWARDS FOR ANY USER - INFORMATIONAL

Description:

Users can call the claim() function from RewardDistributor.sol contract, which transfers the corresponding amount to the user included in the Merkle tree.

However, it has been detected that, since no msg.sender check is performed, any user can call the claim() function and, if correct, account receives the appropriate tokens amount. This does not suppose a risk of rewards stealing since bSECT and lveSECT are minted to account, not to msg.sender, but allows any user to call claim() for any user at any time.

Code Location:

Listing 15: RewardDistributor.sol

```
47     function claim(
48         address account,
49         uint256 totalAmount,
50         bytes32[] calldata merkleProof
51     ) external override {
52         require(merkleRoot != bytes32(0), "MerkleDistributor: No
↳ merkle root set");
53         require(claimed[account] < totalAmount, "MerkleDistributor
↳ : Nothing to claim");
54 
55         bytes32 node = keccak256(bytes.concat(keccak256(abi.encode
↳ (account, totalAmount))));
56 
57         require(
58             MerkleProof.verify(merkleProof, merkleRoot, node),
59             "MerkleDistributor: Invalid proof."
60         );
61 
62         uint256 claimedAmount = totalAmount - claimed[account];
```

```
63          claimed[account] += claimedAmount;
64
65          uint256 bTokenAmount = claimedAmount / 2;
66          uint256 lveTokenAmount = claimedAmount - bTokenAmount;
67          bToken.mintTo(account, bTokenAmount);
68          lveToken.mintTo(account, lveTokenAmount);
69
70          emit Claimed(account, claimedAmount, false);
71      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding a `require` statement in the fashion of `require(account == msg.sender, "Only can claim for yourself")` if timing is relevant when calling `claim()`, or if there could be any disadvantage or harm to users if rewards are claimed at a specific moment.

Remediation Plan:

ACKNOWLEDGED: The Sector Finance team acknowledged this issue as it is considered intended behavior.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the 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.

Results:

bSECT.sol

```
bSECT.mintTo(address,uint256) (contracts/bSECT.sol#29-33) ignores return value by SECT.transferFrom(msg.sender,address(this),amount) (contracts/bSECT.sol#31)
bSECT.convert(uint256) (contracts/bSECT.sol#35-44) ignores return value by SECT.transfer(msg.sender,amount) (contracts/bSECT.sol#42)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Reentrancy in bSECT.convert(uint256) (contracts/bSECT.sol#35-44):
        External calls:
        - underlying.safeTransferFrom(msg.sender,address(this),underlyingAmnt) (contracts/bSECT.sol#41)
        - SECT.transfer(msg.sender,amount) (contracts/bSECT.sol#42)
        Event emitted after the call(s):
        - Convert(msg.sender,amount) (contracts/bSECT.sol#43)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Contract bSECT (contracts/bSECT.sol#8-63) is not in CapWords
Variable bSECT.SECT (contracts/bSECT.sol#11) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

lveSECT.sol

```
lveSECT.mintTo(address,uint256) (contracts/lveSECT.sol#26-30) ignores return value by sect.transferFrom(msg.sender,address(this),amount) (contracts/lveSECT.sol#28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

lveSECT.setVeToken(address) (contracts/lveSECT.sol#20-24) ignores return value by sect.approve(veToken_,type()(uint256).max) (contracts/lveSECT.sol#22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in lveSECT.addValueToLock(uint256) (contracts/lveSECT.sol#45-52):
        External calls:
        - veSECT.increaseAmountFor(msg.sender,amount) (contracts/lveSECT.sol#50)
        Event emitted after the call(s):
        - AddValueToLock(msg.sender,amount) (contracts/lveSECT.sol#51)
Reentrancy in lveSECT.convertToLock(uint256) (contracts/lveSECT.sol#35-41):
        External calls:
        - veSECT.lockFor(msg.sender,amount,expiry) (contracts/lveSECT.sol#39)
        Event emitted after the call(s):
        - ConvertToLock(msg.sender,amount,expiry) (contracts/lveSECT.sol#40)
Reentrancy in lveSECT.setVeToken(address) (contracts/lveSECT.sol#20-24):
        External calls:
        - sect.approve(veToken_,type()(uint256).max) (contracts/lveSECT.sol#22)
        Event emitted after the call(s):
        - SetVeToken(address(veSECT)) (contracts/lveSECT.sol#23)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

lveSECT.addValueToLock(uint256) (contracts/lveSECT.sol#45-52) uses timestamp for comparisons
        Dangerous comparisons:
        - expiry > lockEnd (contracts/lveSECT.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Contract lveSECT (contracts/lveSECT.sol#10-60) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IVotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#33) is too similar to IVotingEscrow.lockFor(address,uint256,uint256).unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable IVotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#8) is too similar to IVotingEscrow.lockFor(address,uint256,uint256).unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

lveSECT.duration (contracts/lveSECT.sol#14) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

RewardDistributor.sol

```
RewardDistributor.constructor(address,address,address,bytes32) (contracts/RewardDistributor.sol#27-34) ignores return value by token.approve(bToken_,type()(uint256).max) (contracts/RewardDistributor.sol#32)
RewardDistributor.constructor(address,address,address,bytes32) (contracts/RewardDistributor.sol#27-34) ignores return value by token.approve(lveToken_,type()(uint256).max) (contracts/RewardDistributor.sol#33)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in RewardDistributor.claim(uint256,address,uint256,bytes32[]) (contracts/RewardDistributor.sol#42-69):
        External calls:
        - bToken.mintTo(account,bTokenAmount) (contracts/RewardDistributor.sol#65)
        - lveToken.mintTo(account,lveTokenAmount) (contracts/RewardDistributor.sol#66)
        Event emitted after the call(s):
        - Claimed(account,claimedAmount,false) (contracts/RewardDistributor.sol#68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Pragma version>=0.5.0 (contracts/interfaces/IRewardDistributor.sol#2) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

SECT.sol

```
Parameter SECT.burn(uint256)._amount (contracts/SECT.sol#21) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

SECT.constructor() (contracts/SECT.sol#9-12) uses literals with too many digits:
        - _mint(address(this),100000000e18) (contracts/SECT.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

AUTOMATED TESTING

VotingEscrow.sol

```
VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance) (contracts/VotingEscrow.sol#241-407) performs a multiplication on the result of a division:
        - slopeDelta = (MULTIPLIER * (block.number - lastPoint.blk)) / (block.timestamp - lastPoint.ts) (contracts/VotingEscrow.sol#318-320)
        - lastPoint.blk = initialLastPoint.blk + (blockSlope * (iterativeTime - initialLastPoint.ts)) / MULTIPLIER (contracts/VotingEscrow.sol#350-353)
VotingEscrow._floorToWeek(uint256) (contracts/VotingEscrow.sol#753-755) performs a multiplication on the result of a division:
        - (_t / WEEK) * WEEK (contracts/VotingEscrow.sol#754)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance) (contracts/VotingEscrow.sol#241-407) uses a dangerous strict equality:
        - _newLocked.end == _oldLocked.end (contracts/VotingEscrow.sol#288)
VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance) (contracts/VotingEscrow.sol#241-407) uses a dangerous strict equality:
        - iterativeTime == block.timestamp (contracts/VotingEscrow.sol#357)
VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance) (contracts/VotingEscrow.sol#241-407) uses a dangerous strict equality:
        - _newLocked.end == _oldLocked.end (contracts/VotingEscrow.sol#394)
VotingEscrow._increaseAmount(address,uint256) (contracts/VotingEscrow.sol#496-537) uses a dangerous strict equality:
        - _delegatee == account (contracts/VotingEscrow.sol#513)
VotingEscrow._supplyAt(VotingEscrow.Point,uint256) (contracts/VotingEscrow.sol#877-914) uses a dangerous strict equality:
        - iterativeTime == _t (contracts/VotingEscrow.sol#899)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance).i (contracts/VotingEscrow.sol#327) is a local variable never initialized
VotingEscrow.checkpoint().empty (contracts/VotingEscrow.sol#411) is a local variable never initialized
VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance).uOldPoint (contracts/VotingEscrow.sol#246) is a local variable never initialized
VotingEscrow._supplyAt(VotingEscrow.Point,uint256).i (contracts/VotingEscrow.sol#882) is a local variable never initialized
VotingEscrow._findUserBlockEpoch(address,uint256).i (contracts/VotingEscrow.sol#708) is a local variable never initialized
VotingEscrow._findBlockEpoch(uint256,uint256).i (contracts/VotingEscrow.sol#765) is a local variable never initialized
VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance).uNewPoint (contracts/VotingEscrow.sol#247) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

VotingEscrow.constructor(address,address,address,string,string)._owner (contracts/VotingEscrow.sol#114) lacks a zero-check on :
        - owner = _owner (contracts/VotingEscrow.sol#133)
VotingEscrow.constructor(address,address,address,string,string)._penaltyRecipient (contracts/VotingEscrow.sol#115) lacks a zero-check on :
        - penaltyRecipient = _penaltyRecipient (contracts/VotingEscrow.sol#134)
VotingEscrow.transferOwnership(address)._addr (contracts/VotingEscrow.sol#160) lacks a zero-check on :
        - owner = _addr (contracts/VotingEscrow.sol#161)
VotingEscrow.updateBlocklist(address,_addr (contracts/VotingEscrow.sol#166) lacks a zero-check on :
        - blocklist = _addr (contracts/VotingEscrow.sol#167)
VotingEscrow.updatePenaltyRecipient(address,_addr (contracts/VotingEscrow.sol#172) lacks a zero-check on :
        - penaltyRecipient = _addr (contracts/VotingEscrow.sol#173)
VotingEscrow.updateLveSEC(address)._addr (contracts/VotingEscrow.sol#205) lacks a zero-check on :
        - LveSEC = _addr (contracts/VotingEscrow.sol#206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in VotingEscrow.collectPenalty() (contracts/VotingEscrow.sol#729-735):
        External calls:
        - token.safeTransfer(recipient,amount) (contracts/VotingEscrow.sol#733)
        Event emitted after the call(s):
        - CollectPenalty(amount,recipient) (contracts/VotingEscrow.sol#734)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

VotingEscrow.forceUndelegate(address) (contracts/VotingEscrow.sol#180-202) uses timestamp for comparisons
        Dangerous comparisons:
        - _delegatee != _addr && value > 0 (contracts/VotingEscrow.sol#194)
VotingEscrow._checkpoint(address,VotingEscrow.LockedBalance,VotingEscrow.LockedBalance) (contracts/VotingEscrow.sol#241-407) uses timestamp for comparisons
        Dangerous comparisons:
        - _addr != address(0) (contracts/VotingEscrow.sol#252)
        - _oldLocked.end > block.timestamp && _oldLocked.delegated > 0 (contracts/VotingEscrow.sol#258)
        - _newLocked.end > block.timestamp && _newLocked.delegated > 0 (contracts/VotingEscrow.sol#264)
        - _newLocked.end != 0 (contracts/VotingEscrow.sol#287)
        - _newLocked.end == _oldLocked.end (contracts/VotingEscrow.sol#288)
        - block.timestamp > lastPoint.ts (contracts/VotingEscrow.sol#317)
        - iterativeTime > block.timestamp (contracts/VotingEscrow.sol#322)
        - lastPoint.bias < 0 (contracts/VotingEscrow.sol#341)
        - lastPoint.slope < 0 (contracts/VotingEscrow.sol#345)
        - iterativeTime == block.timestamp (contracts/VotingEscrow.sol#357)
        - _addr != address(0) (contracts/VotingEscrow.sol#371)
        - lastPoint.slope < 0 (contracts/VotingEscrow.sol#376)
        - lastPoint.bias < 0 (contracts/VotingEscrow.sol#379)
        - _addr != address(0) (contracts/VotingEscrow.sol#387)
        - _oldLocked.end > block.timestamp (contracts/VotingEscrow.sol#391)
        - _newLocked.end == _oldLocked.end (contracts/VotingEscrow.sol#394)
        - _newLocked.end > block.timestamp (contracts/VotingEscrow.sol#399)
        - _newLocked.end > _oldLocked.end (contracts/VotingEscrow.sol#400)
VotingEscrow._lockFor(address,uint256,uint256) (contracts/VotingEscrow.sol#443-470) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.delegatee == account || locked_.amount == 0,Delegated lock) (contracts/VotingEscrow.sol#451)
        - require(bool,string)(unlock_time >= locked_.end,Only increase lock end) (contracts/VotingEscrow.sol#452)
        - require(bool,string)(unlock_time > block.timestamp,Only future lock end) (contracts/VotingEscrow.sol#453)
        - require(bool,string)(unlock_time <= block.timestamp + MAXTIME,Exceeds maxtime) (contracts/VotingEscrow.sol#454)
VotingEscrow._increaseAmount(address,uint256) (contracts/VotingEscrow.sol#496-537) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.amount > 0,No lock) (contracts/VotingEscrow.sol#500)
        - require(bool,string)(locked_.end > block.timestamp,Lock expired) (contracts/VotingEscrow.sol#501)
        - delegatee == account (contracts/VotingEscrow.sol#512)
        - require(bool,string)(locked_.amount > 0,Delegatee has no lock) (contracts/VotingEscrow.sol#525)
        - require(bool,string)(locked_.end > block.timestamp,Delegatee lock expired) (contracts/VotingEscrow.sol#526)
VotingEscrow.increaseUnlockTime(uint256) (contracts/VotingEscrow.sol#545-564) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.amount > 0,No lock) (contracts/VotingEscrow.sol#549)
        - require(bool,string)(locked_.end > block.timestamp,Lock expired) (contracts/VotingEscrow.sol#550)
        - require(bool,string)(unlock_time > locked_.end,Only increase lock end) (contracts/VotingEscrow.sol#551)
        - require(bool,string)(unlock_time <= block.timestamp + MAXTIME,Exceeds maxtime) (contracts/VotingEscrow.sol#552)
        - locked_.delegated > 0 (contracts/VotingEscrow.sol#557)
VotingEscrow.withdraw() (contracts/VotingEscrow.sol#568-592) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.end <= block.timestamp,Lock not expired) (contracts/VotingEscrow.sol#572)
VotingEscrow.delegate(address) (contracts/VotingEscrow.sol#602-628) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.amount > 0,No lock) (contracts/VotingEscrow.sol#611)
        - require(bool,string)(locked_.end > block.timestamp,Lock expired) (contracts/VotingEscrow.sol#612)
        - require(bool,string)(locked_.delegatee != _addr,Already delegated) (contracts/VotingEscrow.sol#613)
        - require(bool,string)(toLocked.amount > 0,Delegatee has no lock) (contracts/VotingEscrow.sol#623)
        - require(bool,string)(toLocked.end > block.timestamp,Delegatee lock expired) (contracts/VotingEscrow.sol#624)
        - require(bool,string)(toLocked.end >= locked_.end,Only delegate to longer lock) (contracts/VotingEscrow.sol#625)
VotingEscrow._unDelegate() (contracts/VotingEscrow.sol#633-648) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.amount > 0,No lock) (contracts/VotingEscrow.sol#636)
        - require(bool,string)(locked_.delegatee != msg.sender,Already undelegated) (contracts/VotingEscrow.sol#637)
        - locked_.end < fromLocked.end (contracts/VotingEscrow.sol#643)
VotingEscrow._delegate(address,VotingEscrow.LockedBalance,int128,VotingEscrow.LockAction) (contracts/VotingEscrow.sol#651-670) uses timestamp for comparisons
        Dangerous comparisons:
        - newLocked.amount > 0 (contracts/VotingEscrow.sol#666)
VotingEscrow.quitLock() (contracts/VotingEscrow.sol#679-708) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(locked_.end > block.timestamp,Lock expired) (contracts/VotingEscrow.sol#683)
VotingEscrow._findBlockEpoch(uint256,uint256) (contracts/VotingEscrow.sol#768-778) uses timestamp for comparisons
        Dangerous comparisons:
        - pointHistory[mid].blk <= _block (contracts/VotingEscrow.sol#768)
VotingEscrow.balanceOf(address) (contracts/VotingEscrow.sol#806-821) uses timestamp for comparisons
        Dangerous comparisons:
        - lastPoint.bias < 0 (contracts/VotingEscrow.sol#817)
VotingEscrow.balanceOfAt(address,uint256) (contracts/VotingEscrow.sol#827-872) uses timestamp for comparisons
        Dangerous comparisons:
        - dBlock != 0 (contracts/VotingEscrow.sol#860)
        - upoint.bias >= 0 (contracts/VotingEscrow.sol#867)
VotingEscrow._supplyAt(VotingEscrow.Point,uint256) (contracts/VotingEscrow.sol#877-914) uses timestamp for comparisons
        Dangerous comparisons:
        - iterativeTime > _t (contracts/VotingEscrow.sol#886)
        - iterativeTime == _t (contracts/VotingEscrow.sol#899)
        - lastPoint.bias < 0 (contracts/VotingEscrow.sol#910)
VotingEscrow.totalSupplyAt(uint256) (contracts/VotingEscrow.sol#927-955) uses timestamp for comparisons
        Dangerous comparisons:
        - point.blk > _blockNumber (contracts/VotingEscrow.sol#936)
        - point.blk != pointNext.blk (contracts/VotingEscrow.sol#943)
        - point.blk != block.number (contracts/VotingEscrow.sol#948)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Parameter VotingEscrow.transferOwnership(address)._addr (contracts/VotingEscrow.sol#160) is not in mixedCase
Parameter VotingEscrow.updateBlocklist(address,_addr (contracts/VotingEscrow.sol#166) is not in mixedCase
Parameter VotingEscrow.updatePenaltyRecipient(address,_addr (contracts/VotingEscrow.sol#172) is not in mixedCase
Parameter VotingEscrow.forceUndelegate(address)._addr (contracts/VotingEscrow.sol#180) is not in mixedCase
Parameter VotingEscrow.updateLveSEC(address)._addr (contracts/VotingEscrow.sol#205) is not in mixedCase
Parameter VotingEscrow.lockedEnd(address)._addr (contracts/VotingEscrow.sol#217) is not in mixedCase
Parameter VotingEscrow.getLastUserPoint(address)._addr (contracts/VotingEscrow.sol#227) is not in mixedCase
Parameter VotingEscrow.createLock(uint256,uint256)._value (contracts/VotingEscrow.sol#422) is not in mixedCase
Parameter VotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#423) is not in mixedCase
Parameter VotingEscrow.lockFor(address,uint256,uint256)._account (contracts/VotingEscrow.sol#436) is not in mixedCase
Parameter VotingEscrow.lockFor(address,uint256,uint256)._value (contracts/VotingEscrow.sol#437) is not in mixedCase
Parameter VotingEscrow.lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#438) is not in mixedCase
Parameter VotingEscrow.increaseAmountFor(address,uint256)._value (contracts/VotingEscrow.sol#481) is not in mixedCase
Parameter VotingEscrow.increaseAmount(uint256)._value (contracts/VotingEscrow.sol#493) is not in mixedCase
Parameter VotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/VotingEscrow.sol#543) is not in mixedCase
Parameter VotingEscrow.delegate(address)._addr (contracts/VotingEscrow.sol#602) is not in mixedCase
Parameter VotingEscrow.balanceOf(address)._owner (contracts/VotingEscrow.sol#806) is not in mixedCase
Parameter VotingEscrow.balanceOfAt(address,uint256)._owner (contracts/VotingEscrow.sol#828) is not in mixedCase
Parameter VotingEscrow.balanceOfAt(address,uint256)._blockNumber (contracts/VotingEscrow.sol#829) is not in mixedCase
Parameter VotingEscrow.totalSupplyAt(uint256)._blockNumber (contracts/VotingEscrow.sol#977) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable VotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#423) is too similar to IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable IVotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#33) is too similar to IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable VotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/VotingEscrow.sol#545) is too similar to IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable VotingEscrow._lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#443) is too similar to IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable IVotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#8) is too similar to IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable VotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/VotingEscrow.sol#545) is too similar to VotingEscrow.increaseUnlockTime(uint256)._unlock_time (contracts/VotingEscrow.sol#547)
Variable VotingEscrow._lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#443) is too similar to VotingEscrow.increaseUnlockTime(uint256)._unlock_time (contracts/VotingEscrow.sol#547)
Variable VotingEscrow._lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#443) is too similar to VotingEscrow._lockFor(address,uint256,uint256)._unlock_time (contracts/VotingEscrow.sol#444)
Variable IVotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#33) is too similar to IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14)
Variable VotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/VotingEscrow.sol#545) is too similar to VotingEscrow._lockFor(address,uint256,uint256)._unlock_time (contracts/VotingEscrow.sol#444)
Variable VotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#423) is too similar to VotingEscrow._lockFor(address,uint256,uint256)._unlock_time (contracts/VotingEscrow.sol#444)
Variable IVotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#33) is too similar to VotingEscrow.increaseUnlockTime(uint256).unlock_time (contracts/VotingEscrow.sol#547)
Variable VotingEscrow.lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#438) is too similar to VotingEscrow.increaseUnlockTime(uint256).unlock_time (contracts/VotingEscrow.sol#547)
Variable IVotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#8) is too similar to VotingEscrow.increaseUnlockTime(uint256).unlock_time (contracts/VotingEscrow.sol#547)
Variable VotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#423) is too similar to VotingEscrow._lockFor(address,uint256,uint256).unlock_time (contracts/VotingEscrow.sol#444)
Variable VotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#423) is too similar to VotingEscrow.increaseUnlockTime(uint256).unlock_time (contracts/VotingEscrow.sol#547)
Variable VotingEscrow.lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#438) is too similar to VotingEscrow.increaseUnlockTime(uint256).unlock_time (contracts/VotingEscrow.sol#547)
Variable IVotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#8) is too similar to VotingEscrow._lockFor(address,uint256,uint256).unlock_time (contracts/VotingEscrow.sol#444)
Variable VotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#423) is too similar to VotingEscrow._lockFor(address,uint256,uint256).unlock_time (contracts/VotingEscrow.sol#444)
Variable IVotingEscrow.increaseUnlockTime(uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#33) is too similar to VotingEscrow._lockFor(address,uint256,uint256).unlock_time (contracts/VotingEscrow.sol#444)
Variable VotingEscrow.lockFor(address,uint256,uint256)._unlockTime (contracts/VotingEscrow.sol#438) is too similar to VotingEscrow._lockFor(address,uint256,uint256).unlock_time (contracts/VotingEscrow.sol#444)
Variable IVotingEscrow.createLock(uint256,uint256)._unlockTime (contracts/interfaces/IVotingEscrow.sol#8) is too similar to VotingEscrow._lockFor(address,uint256,uint256).unlock_time (contracts/VotingEscrow.sol#444)
Variable IVotingEscrow.lockFor(address,uint256,uint256)._unlockTime_ (contracts/interfaces/IVotingEscrow.sol#14) is too similar to VotingEscrow.increaseUnlockTime(uint256).unlock_time (contracts/VotingEscrow.sol#547)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

VotingEscrow.decimals (contracts/VotingEscrow.sol#79) should be immutable
VotingEscrow.name (contracts/VotingEscrow.sol#77) should be immutable
VotingEscrow.symbol (contracts/VotingEscrow.sol#78) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

- Issues found by Slither are either already reported or false positives.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits 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 smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

bSECT.sol

Report for contracts/bSECT.sol
https://dashboard.mythx.io/#/console/analyses/09b57ad6-8716-424f-a002-ef43afb5eebc

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

lveSECT.sol

Report for contracts/lveSECT.sol
https://dashboard.mythx.io/#/console/analyses/65826685-0379-44d7-9092-648f48b236bd

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

RewardDistributor.sol

Report for contracts/RewardDistributor.sol
https://dashboard.mythx.io/#/console/analyses/07dc42bc-8035-49b6-8f8f-c1949f14b2de

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
59	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
60	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+=" discovered
63	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
64	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

SECT.sol

Report for contracts/SECT.sol
https://dashboard.mythx.io/#/console/analyses/25b07abc-f5f6-4baf-b0db-3de476de75d5

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

VotingEscrow.sol

Report for contracts/VotingEscrow.sol
https://dashboard.mythx.io/#/console/analyses/cfbcdeba-37ed-4efe-a27f-5f88f73ba5be

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
70	(SWC-110) Assert Violation	Unknown	Public state variable with array type causing reacheable exception by default.
71	(SWC-110) Assert Violation	Unknown	Public state variable with array type causing reacheable exception by default.
121	(SWC-110) Assert Violation	Unknown	Out of bounds array access
125	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
233	(SWC-110) Assert Violation	Unknown	Out of bounds array access
259	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
261	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
262	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
265	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
267	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
268	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
276	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
278	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
279	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
279	(SWC-110) Assert Violation	Unknown	Out of bounds array access
300	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
303	(SWC-110) Assert Violation	Unknown	Out of bounds array access
319	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
319	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
319	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
319	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
320	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
330	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
337	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
337	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
338	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
339	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
351	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
352	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
352	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
352	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
356	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
358	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
361	(SWC-110) Assert Violation	Unknown	Out of bounds array access

364	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
374	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
374	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
375	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
375	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
385	(SWC-110) Assert Violation	Unknown	Out of bounds array access
393	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
395	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
401	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
454	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
456	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
461	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
463	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
503	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
516	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
517	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
521	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
528	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
552	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
576	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
581	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
659	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
662	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
687	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
691	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
702	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
702	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
703	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
704	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
724	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
724	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
724	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
754	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
754	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
767	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
767	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
768	(SWC-110) Assert Violation	Unknown	Out of bounds array access

771	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
771	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
774	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
790	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
790	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
791	(SWC-110) Assert Violation	Unknown	Out of bounds array access
794	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
794	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
797	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
813	(SWC-110) Assert Violation	Unknown	Out of bounds array access
815	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
816	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
816	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
831	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
838	(SWC-110) Assert Violation	Unknown	Out of bounds array access
843	(SWC-110) Assert Violation	Unknown	Out of bounds array access
851	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
851	(SWC-110) Assert Violation	Unknown	Out of bounds array access
852	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
853	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
855	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
855	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
856	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
861	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
861	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
861	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
861	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
866	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
866	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
883	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
897	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
898	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
898	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
902	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
906	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
920	(SWC-110) Assert Violation	Unknown	Out of bounds array access
928	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
933	(SWC-110) Assert Violation	Unknown	Out of bounds array access
942	(SWC-110) Assert Violation	Unknown	Out of bounds array access
942	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
945	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
945	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
945	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
946	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
948	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
950	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
950	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
950	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
951	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
951	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
954	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered

- Issues found by MythX are either already reported or false positives.

THANK YOU FOR CHOOSING

// HALBORN