// HALBORN

EGL

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 27th, 2021 - July 7th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/27/2021	Gokberk Gulgun
0.5	Document Updates	07/02/2021	Gokberk Gulgun
1.0	Final Version	07/07/2021	Gabi Urrutia
1.1	Remediation Plan	07/21/2021	Gokberk Gulgun
1.1	Remediation Review	07/26/2021	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

EGL engaged Halborn to conduct a security assessment on their Smart contract beginning on June 27th, 2021 and July 7th, 2021.

The security assessment was scoped to the smart contract repository. An audit of the security risk and implications regarding the changes introduced by the development team at EGL prior to its production release shortly following the assessments deadline.

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

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks 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 to achieve the following:

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

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit.While manual testing is recommended

to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Symbolic Execution / EVM bytecode security assessment (Manticore)
- Testnet deployment (Truffle, Ganache)

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

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

1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

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

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

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

The security assessment was scoped to the smart contracts :
- EglContract.sol
- EglToken.sol

Specific commit ID of the contract:
1c02789e5f0c89a12f2e7a505a6c07da5c37f8f1

Remediated commit ID of the contract :
07889444d8b40ea2cd5845850b6008c995d8c144

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	6	3

LIKELIHOOD

IMPACT

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - DIVIDE BEFORE MULTIPLY	Low	ACKNOWLEDGED: 07/21/2021
HAL02 - LACK OF MULTIPLE VOTING CHECK	Low	SOLVED: 07/21/2021
HAL03 - LACK OF ACCESS CONTROL ON THE MANAGEMENT PARAMETERS	Low	NOT APPLICABLE: 07/21/2021
HAL04 - USE OF BLOCK.TIMESTAMP	Low	NOT APPLICABLE: 07/21/2021
HAL05 - MISSING EVENT HANDLER	Low	SOLVED: 07/21/2021
HAL06 - IGNORED RETURN VALUES	Low	SOLVED: 07/21/2021
HAL07 - USE OF ASSERT	Informational	SOLVED: 07/21/2021
HAL08 - MISSING RE-ENTRANCY PROTECTION	Informational	SOLVED: 07/21/2021
HAL09 - BLOCK TIMESTAMP ALIAS USAGE	Informational	NOT APPLICABLE: 07/21/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) DIVIDE BEFORE MULTIPLY - LOW

Description:

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision. In this audit, there are multiple instances found where division is being performed before multiplication operation in the EglContract.sol.

Code Location:

EglContract.sol Line #968

Listing 1: EglContract.sol (Lines)

```
968 proximityRewardPercent = uint(actualDelta.mul(int(
        DECIMAL_PRECISION)).div(eglDelta)).mul(75);
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider doing multiplication operation before division to prevail precision in the values in non floating data type.

Remediation Plan:

ACKNOWLEDGED: actualDelta is multiplied by DECIMAL_PRECISION before being divided by eglDelta, then the precision is maintained by EGL Team.

3.2 (HAL-02) LACK OF MULTIPLE VOTING CHECK - LOW

Description:

In the vote progress, the pre-condition checks are not applied on the function. Although _internalVote function prevents to multiple votes, the initial checks should complete at the beginning of the function.

Code Location:

EglContract.sol Line #968

Listing 2: EglContract.sol (Lines)

```
477     function vote(
478         uint _gasTarget,
479         uint _eglAmount,
480         uint8 _lockupDuration
481     )
482         external
483         whenNotPaused
484     {
485         require(_eglAmount >= 1 ether, "EGL:AMNT_TOO_LOW");
486         require(_eglAmount <= eglToken.balanceOf(msg.sender), "EGL
                :INSUFFICIENT_EGL_BALANCE");
487         require(eglToken.allowance(msg.sender, address(this)) >=
                _eglAmount, "EGL:INSUFFICIENT_ALLOWANCE");
488         if (block.timestamp > currentEpochStartDate.add(
                epochLength))
489             tallyVotes();
490
491         eglToken.transferFrom(msg.sender, address(this),
                _eglAmount);
492         _internalVote(
493             msg.sender,
494             _gasTarget,
495             _eglAmount,
496             _lockupDuration,
497             0
```

```
498         );
499     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

The initial checks should complete through ‘require()‘‘ function. The sample solution can be seen below.

Listing 3: EglContract.sol (Lines 9)

```
1     function vote(
2         uint _gasTarget,
3         uint _eglAmount,
4         uint8 _lockupDuration
5     )
6         external
7         whenNotPaused
8     {
9     require(voters[msg.sender].tokensLocked == 0, "EGL:
          ALREADY_VOTED");
10         require(_eglAmount >= 1 ether, "EGL:AMNT_TOO_LOW");
11         require(_eglAmount <= eglToken.balanceOf(msg.sender), "EGL
              :INSUFFICIENT_EGL_BALANCE");
12         require(eglToken.allowance(msg.sender, address(this)) >=
              _eglAmount, "EGL:INSUFFICIENT_ALLOWANCE");
13         if (block.timestamp > currentEpochStartDate.add(
              epochLength))
14             tallyVotes();
15
16         eglToken.transferFrom(msg.sender, address(this),
              _eglAmount);
17         _internalVote(
18             msg.sender,
19             _gasTarget,
20             _eglAmount,
21             _lockupDuration,
```

```
22              0
23          );
24      }
```

Remediation Plan:

SOLVED: Validations were completed in the internal function by EGL Team.

3.3 (HAL-03) LACK OF ACCESS CONTROL ON THE MANAGEMENT PARAMETERS - LOW

Description:

In the MockEglGenesis contract, canWithdraw and canContribute variable is defined as a public function. These function could be called by anyone. Although, the contract named as MockEglGenesis, the modifier should add at the beginning of the function.

Code Location:

MockEglGenesis.sol Line #33-39

Listing 4: MockEglGenesis.sol (Lines 33,37)

```
33      function setCanContribute(bool _canContribute) external {
34          canContribute = _canContribute;
35      }
36
37      function setCanWithdraw(bool _canWithdraw) external {
38          canWithdraw = _canWithdraw;
39      }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommend to add onlyOwner modifier at the beginning of the function. Example remediated code can be seen below.

Listing 5: RemediatedMockEglGenesis.sol (Lines 33,37)

```
33 function setCanContribute(bool _canContribute) external onlyOwner
       {
34          canContribute = _canContribute;
35      }
36
37 function setCanWithdraw(bool _canWithdraw) external onlyOwner {
38          canWithdraw = _canWithdraw;
39      }
```

Remediation Plan:

NOT APPLICABLE: The mock-up contracts are added only for testing purpose.

3.4 (HAL-04) USE OF BLOCK.TIMESTAMP - LOW

Description:

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

Code Location:

EglContract.sol Line #33-39

Listing 6: EglContract.sol (Lines 392)

```
380     function claimSupporterEgls(uint _gasTarget, uint8
            _lockupDuration) external whenNotPaused {
381         require(remainingSupporterBalance > 0, "EGL:
                SUPPORTER_EGLS_DEPLETED");
382         require(remainingBptBalance > 0, "EGL:BPT_BALANCE_DEPLETED
                ");
383         require(
384             eglGenesis.canContribute() == false && eglGenesis.
                    canWithdraw() == false,
385             "EGL:GENESIS_LOCKED"
386         );
387         require(supporters[msg.sender].claimed == 0, "EGL:
                ALREADY_CLAIMED");
388
389         (uint contributionAmount, uint cumulativeBalance, ,) =
                eglGenesis.contributors(msg.sender);
390         require(contributionAmount > 0, "EGL:NOT_CONTRIBUTED");
391
392         if (block.timestamp > currentEpochStartDate.add(
                epochLength))
393             tallyVotes();
394         ...
```

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

NOT APPLICABLE: the EGL Team considers safe the usage of block.timestamp because 900 seconds of drift from miners is preferable to other options. Calculating time from the block could be wrong if there is a fork or upgrade - timestamps are less vulnerable to a change in block duration that could occur with Ethereum 2.0 upgrades or hard forks. Use of oracles would create a dependency on the health of a third party service and potentially incur additional fees.

3.5 (HAL-05) MISSING EVENT HANDLER - LOW

Description:

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

EglContract.sol Line #~542

Listing 7: EglContract.sol (Lines 542)

```
542     function withdraw() external whenNotPaused {
543         require(voters[msg.sender].tokensLocked > 0, "EGL:
                NOT_VOTED");
544         require(block.timestamp > voters[msg.sender].releaseDate,
                "EGL:NOT_RELEASE_DATE");
545         eglToken.transfer(msg.sender, _internalWithdraw(msg.sender
                ));
546     }
```

EglContract.sol Line #~640

Listing 8: EglContract.sol (Lines)

```
640     function pauseEgl() external onlyOwner whenNotPaused {
641         _pause();
642     }
643
644     /**
645      * @notice Owner only function to unpause contract
646      */
647     function unpauseEgl() external onlyOwner whenPaused {
648         _unpause();
649     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

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

Remediation Plan:

SOLVED: The Withdraw event is now emitted from the internal withdraw function, as well as Pause and Unpause events are emitted from the parent functions.

3.6 (HAL-06) IGNORED RETURN VALUES - LOW

Description:

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

Code Location:

EglContract.sol Line #~491,522,545,560,631,915

Listing 9: EglContract.sol (Lines 491)

```
491   eglToken.transferFrom(msg.sender, address(this), _eglAmount);
492
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Add a return value check to avoid an unexpected crash of the contract. Return value checks provide better exception handling. As an other solution, Use SafeERC20 on the both networks when possible and ensure that the transfer/transferFrom return value is checked. A custom function named safeTransferFrom can be implemented that checks the return value of the transferFrom function and makes sure that the funds were transferred.

Remediation Plan:

SOLVED: The return values are now checked on the transferFrom function.

Listing 10: EglContract.sol (Lines)

```
491         bool success = eglToken.transferFrom(msg.sender, address(
                this), _eglAmount);
492         require(success, "EGL:TOKEN_TRANSFER_FAILED");
```

3.7 (HAL-07) USE OF ASSERT - INFORMATIONAL

Description:

In the solidity, assert() and require() functions are used for protecting contract on the unexceptional behaviours. With the assert() function, it is not possible to evaluate return value.

Code Location:

EglContract.sol Line #802

Listing 11: EglContract.sol (Lines 802)

```
795     function _internalVote(
796         address _voter,
797         uint _gasTarget,
798         uint _eglAmount,
799         uint8 _lockupDuration,
800         uint _releaseTime
801     ) internal {
802         assert(_voter != address(0));
803     ...
804     }
```

EglContract.sol Line #1049

Listing 12: EglContract.sol (Lines 1049)

```
1039     function _calculateCurrentPoolTokensDue(
1040         uint _currentEgl,
1041         uint _firstEgl,
1042         uint _lastEgl,
1043         uint _totalPoolTokens
1044     )
1045         internal
1046         pure
1047         returns (uint poolTokensDue)
```

```
1048     {
1049         assert(_firstEgl < _lastEgl);
1050 
1051         if (_currentEgl < _firstEgl)
1052             return 0;
1053 
1054         uint eglsReleased = (_currentEgl.umin(_lastEgl)).sub(
             _firstEgl);
1055         poolTokensDue = _totalPoolTokens
1056             .mul(eglsReleased)
1057             .div(
1058                 _lastEgl.sub(_firstEgl)
1059             );
1060     }
```

EglContract.sol Line #1077

Listing 13: EglContract.sol (Lines 1077)

```
1069     function _calculateBonusEglsDue(
1070         uint _firstEgl,
1071         uint _lastEgl
1072     )
1073         internal
1074         pure
1075         returns (uint bonusEglsDue)
1076     {
1077         assert(_firstEgl < _lastEgl);
1078 
1079         bonusEglsDue = (_lastEgl.div(DECIMAL_PRECISION)**4)
1080             .sub(_firstEgl.div(DECIMAL_PRECISION)**4)
1081             .mul(DECIMAL_PRECISION)
1082             .div(
1083                 (81/128)*(10**27)
1084             );
1085     }
```

EglContract.sol Line #1107

Listing 14: EglContract.sol (Lines 1107)

```
1097     function _calculateVoterReward(
1098         address _voter,
1099         uint16 _currentEpoch,
1100         uint16 _voterEpoch,
1101         uint8 _lockupDuration,
1102         uint _voteWeight
1103     )
1104         internal
1105         returns(uint rewardsDue)
1106     {
1107         assert(_voter != address(0));
1108         ...
1109     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use require() function instead of assert() function. This change will improve code readability. The example code snippet can be seen below.

Listing 15: EglContract.sol (Lines 802)

```
795     function _internalVote(
796         address _voter,
797         uint _gasTarget,
798         uint _eglAmount,
799         uint8 _lockupDuration,
800         uint _releaseTime
801     ) internal {
802         require(_voter != address(0),"EGL:
                VOTER_COULD_NOT_BE_ZERO_ADDRESS");
803     ...
804     }
```

Remediation Plan:

SOLVED: The assert() function is replaced by require() function on the related statements.

Listing 16: EglContract.sol (Lines 802)

```
114     function _calculateVoterReward(
115         address _voter,
116         uint16 _currentEpoch,
117         uint16 _voterEpoch,
118         uint8 _lockupDuration,
119         uint _voteWeight
120     )
121         internal
122         returns(uint rewardsDue)
123     {
124         require(_voter != address(0), "EGL:VOTER_ADDRESS_0");
```

3.8 (HAL-08) MISSING RE-ENTRANCY PROTECTION - INFORMATIONAL

Description:

To protect against cross-function reentrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the vote and re-vote functions with a recursive call. OpenZeppelin has it's own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against Reentrancy attacks.

Code Location:

Listing 17: EglContract.sol (Lines 491)

```
477     function vote(
478         uint _gasTarget,
479         uint _eglAmount,
480         uint8 _lockupDuration
481     )
482         external
483         whenNotPaused
484     {
485         require(_eglAmount >= 1 ether, "EGL:AMNT_TOO_LOW");
486         require(_eglAmount <= eglToken.balanceOf(msg.sender), "EGL
                :INSUFFICIENT_EGL_BALANCE");
487         require(eglToken.allowance(msg.sender, address(this)) >=
                _eglAmount, "EGL:INSUFFICIENT_ALLOWANCE");
488         if (block.timestamp > currentEpochStartDate.add(
                epochLength))
489             tallyVotes();
490
491         eglToken.transferFrom(msg.sender, address(this),
                _eglAmount);
492         _internalVote(
493             msg.sender,
494             _gasTarget,
495             _eglAmount,
496             _lockupDuration,
```

```
497             0
498         );
499     }
```

Listing 18: EglContract.sol (Lines 522)

```
509     function reVote(
510         uint _gasTarget,
511         uint _eglAmount,
512         uint8 _lockupDuration
513     )
514         external
515         whenNotPaused
516     {
517         require(voters[msg.sender].tokensLocked > 0, "EGL:
                NOT_VOTED");
518         if (_eglAmount > 0) {
519             require(_eglAmount >= 1 ether, "EGL:AMNT_TOO_LOW");
520             require(_eglAmount <= eglToken.balanceOf(msg.sender),
                    "EGL:INSUFFICIENT_EGL_BALANCE");
521             require(eglToken.allowance(msg.sender, address(this))
                    >= _eglAmount, "EGL:INSUFFICIENT_ALLOWANCE");
522             eglToken.transferFrom(msg.sender, address(this),
                    _eglAmount);
523         }
524         if (block.timestamp > currentEpochStartDate.add(
                epochLength))
525             tallyVotes();
526
527         uint originalReleaseDate = voters[msg.sender].releaseDate;
528         _eglAmount = _eglAmount.add(_internalWithdraw(msg.sender))
                ;
529         _internalVote(
530             msg.sender,
531             _gasTarget,
532             _eglAmount,
533             _lockupDuration,
534             originalReleaseDate
535         );
536         emit ReVote(msg.sender, _gasTarget, _eglAmount, now);
537     }
```

Recommendation:

In the EglContract.sol contract, the vote() and reVote() functions are missing nonReentrant guard. Use the nonReentrant modifier to avoid introducing future vulnerabilities.

Remediation Plan:

SOLVED: The re-entrancy protection was added into the function.

Listing 19: EglContract.sol (Lines 522)

```
509 function reVote(uint _gasTarget,uint _eglAmount, uint8
       _lockupDuration) external whenNotPaused nonReentrant
510 function vote(uint _gasTarget,uint _eglAmount, uint8
       _lockupDuration) external whenNotPaused nonReentrant
```

3.9 (HAL-09) BLOCK TIMESTAMP ALIAS USAGE - INFORMATIONAL

Description:

During a manual static review, the Halborn Team noticed the use of now. The global variable now is deprecated after the pragma version v0.7.0.

Solidity Pragma Version 0.7.0 - Now Deprecated

Code Location:

Listing 20: EglContract.sol (Lines 763)

```
758     function addSeedAccount(address _seedAccount, uint _seedAmount
            ) public onlyOwner {
759         require(_seedAmount <= remainingSeederBalance, "EGL:
                INSUFFICIENT_SEED_BALANCE");
760         require(seeders[_seedAccount] == 0, "EGL:ALREADY_SEEDER");
761         require(voters[_seedAccount].tokensLocked == 0, "EGL:
                ALREADY_HAS_VOTE");
762         require(eglToken.balanceOf(_seedAccount) == 0, "EGL:
                ALREADY_HAS_EGLS");
763         require(now < firstEpochStartDate.add(
                minLiquidityTokensLockup), "EGL:SEED_PERIOD_PASSED");
764         (uint contributorAmount,,,) = eglGenesis.contributors(
                _seedAccount);
765         require(contributorAmount == 0, "EGL:IS_CONTRIBUTOR");
766
767         remainingSeederBalance = remainingSeederBalance.sub(
                _seedAmount);
768         remainingDaoBalance = remainingDaoBalance.sub(_seedAmount)
                ;
769         seeders[_seedAccount] = _seedAmount;
770         emit SeedAccountAdded(
771             _seedAccount,
772             _seedAmount,
773             remainingSeederBalance,
774             now
775         );
776     }
```

Recommendation:

Use block.number instead of block.timestamp or now reduce the influence of miners. If it is not possible to use block.number, now should replace with block.timestamp.

Remediation Plan:

NOT APPLICABLE: now was replaced by block.timestamp function. EGL Team considers safe the usage of block.timestamp because 900 seconds of drift from miners is preferable to other options. Calculating time from the block could be wrong if there is a fork or upgrade - timestamps are less vulnerable to a change in block duration that could occur with Ethereum 2.0 upgrades or hard forks. the use of oracles would create a dependency on the health of a third party service and potentially incur additional fees.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. 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:

```
INFO:Detectors:
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#74) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
ERC20CappedUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20CappedUpgradeable.sol#51) shadows:
        - ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#312)
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#312) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
```

```
EglContract.tallyVotes() (contracts/EglContract.sol#656-750) performs a multiplication on the result of a division:
        -actualThreshold = votingThreshold.add((DECIMAL_PRECISION.mul(20).div(WEEKS_IN_YEAR.mul(2))).mul(currentEpoch.sub(WEEKS_IN_YEAR.sub(1)))) (contracts/EglContract.sol#665-668)
EglContract._calculateBlockReward(int256,int256,int256) (contracts/EglContract.sol#940-991) performs a multiplication on the result of a division:
        -blockReward = totalRewardPercent.mul(remainingPoolReward.div(5000000)).div(DECIMAL_PRECISION).div(100) (contracts/EglContract.sol#975-977)
EglContract._calculateSerializedEgl(uint256,uint256,uint256) (contracts/EglContract.sol#1001-1029) performs a multiplication on the result of a division:
        -serializedEgl = ((timePassedPercentage.div(10 ** 8)) ** 4).mul(_maxEglSupply.div(DECIMAL_PRECISION)).mul(10 ** 8).div((10 ** 10) ** 3) (contracts/EglContract.sol#1016-1019)
EglContract._calculateSerializedEgl(uint256,uint256,uint256) (contracts/EglContract.sol#1001-1029) performs a multiplication on the result of a division:
        -timePassedPercentage = _timeSinceOrigin.sub(_timeLocked).mul(DECIMAL_PRECISION).div(epochLength.mul(WEEKS_IN_YEAR).sub(_timeLocked)) (contracts/EglContract.sol#1008-1013)
        -serializedEglCalculated(currentEpoch,_timeSinceOrigin,timePassedPercentage.mul(100),serializedEgl,_maxEglSupply,now) (contracts/EglContract.sol#1021-1028)
EglContract._calculateBonusEglsDue(uint256,uint256) (contracts/EglContract.sol#1070-1086) performs a multiplication on the result of a division:
        -bonusEglsDue = (_lastEgl.div(DECIMAL_PRECISION) ** 4).sub(_firstEgl.div(DECIMAL_PRECISION) ** 4).mul(DECIMAL_PRECISION).div((81 / 128) * (10 ** 27)) (contracts/EglContract.sol#1080-1085)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
EglContract._internalVote(address,uint256,uint256,uint8,uint256) (contracts/EglContract.sol#796-850) uses a dangerous strict equality:
        - require(bool,string)(voters[_voter].tokensLocked == 0,EGL:ALREADY_VOTED) (contracts/EglContract.sol#805)
EglContract.addSeedAccount(address,uint256) (contracts/EglContract.sol#759-777) uses a dangerous strict equality:
        - require(bool,string)(eglToken.balanceOf(_seedAccount) == 0,EGL:ALREADY_HAS_EGLS) (contracts/EglContract.sol#763)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Contract locking ether found in :
        Contract EglContract (contracts/EglContract.sol#19-1152) has payable functions:
         - EglContract.receive() (contracts/EglContract.sol#266-268)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

AUTOMATED TESTING

```
INFO:Detectors:
Reentrancy in EglContract._issueCreatorRewards(uint256) (contracts/EglContract.sol#906-930):
        External calls:
        - eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
        State variables written after the call(s):
        - lastSerializedEgl = serializedEgl (contracts/EglContract.sol#929)
        - remainingCreatorReward = remainingCreatorReward.sub(creatorRewardForEpoch) (contracts/EglContract.sol#917)
Reentrancy in EglContract.claimSeederEgls(uint256,uint8) (contracts/EglContract.sol#450-469):
        External calls:
        - tallyVotes() (contracts/EglContract.sol#453)
                - eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
        State variables written after the call(s):
        - _internalVote(msg.sender,_gasTarget,seedAmount,_lockupDuration,releaseDate) (contracts/EglContract.sol#462-468)
                - epochGasLimitSum = epochGasLimitSum.add(int256(block.gaslimit)) (contracts/EglContract.sol#815)
        - _internalVote(msg.sender,_gasTarget,seedAmount,_lockupDuration,releaseDate) (contracts/EglContract.sol#462-468)
                - epochVoteCount = epochVoteCount.add(1) (contracts/EglContract.sol#816)
        - _internalVote(msg.sender,_gasTarget,seedAmount,_lockupDuration,releaseDate) (contracts/EglContract.sol#462-468)
                - gasTargetSum[i] = gasTargetSum[i].add(_gasTarget.mul(voteWeight)) (contracts/EglContract.sol#831)
        - delete seeders[msg.sender] (contracts/EglContract.sol#456)
        - tokensInCirculation = tokensInCirculation.add(seedAmount) (contracts/EglContract.sol#458)
        - _internalVote(msg.sender,_gasTarget,seedAmount,_lockupDuration,releaseDate) (contracts/EglContract.sol#462-468)
                - voteWeightsSum[i] = voteWeightsSum[i].add(voteWeight) (contracts/EglContract.sol#830)
        - _internalVote(msg.sender,_gasTarget,seedAmount,_lockupDuration,releaseDate) (contracts/EglContract.sol#462-468)
                - votesTotal[i] = votesTotal[i].add(_eglAmount) (contracts/EglContract.sol#834)
Reentrancy in EglContract.claimSupporterEgls(uint256,uint8) (contracts/EglContract.sol#381-441):
        External calls:
        - tallyVotes() (contracts/EglContract.sol#394)
                - eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
        State variables written after the call(s):
        - _internalVote(msg.sender,_gasTarget,bonusEglsDue,_lockupDuration,firstEpochStartDate.add(epochLength.mul(WEEKS_IN_YEAR))) (contracts/EglContract.sol#434-440)
                - epochGasLimitSum = epochGasLimitSum.add(int256(block.gaslimit)) (contracts/EglContract.sol#815)
        - _internalVote(msg.sender,_gasTarget,bonusEglsDue,_lockupDuration,firstEpochStartDate.add(epochLength.mul(WEEKS_IN_YEAR))) (contracts/EglContract.sol#434-440)
                - epochVoteCount = epochVoteCount.add(1) (contracts/EglContract.sol#816)
        - _internalVote(msg.sender,_gasTarget,bonusEglsDue,_lockupDuration,firstEpochStartDate.add(epochLength.mul(WEEKS_IN_YEAR))) (contracts/EglContract.sol#434-440)
                - gasTargetSum[i] = gasTargetSum[i].add(_gasTarget.mul(voteWeight)) (contracts/EglContract.sol#831)
        - remainingBptBalance = remainingBptBalance.sub(poolTokensDue) (contracts/EglContract.sol#411)
        - remainingSupporterBalance = remainingSupporterBalance.sub(bonusEglsDue) (contracts/EglContract.sol#410)
        - _supporter.claimed = 1 (contracts/EglContract.sol#415)
        - _supporter.poolTokens = poolTokensDue (contracts/EglContract.sol#416)
        - _supporter.firstEgl = firstEgl (contracts/EglContract.sol#417)
        - _supporter.lastEgl = lastEgl (contracts/EglContract.sol#418)
        - tokensInCirculation = tokensInCirculation.add(bonusEglsDue) (contracts/EglContract.sol#412)
        - _internalVote(msg.sender,_gasTarget,bonusEglsDue,_lockupDuration,firstEpochStartDate.add(epochLength.mul(WEEKS_IN_YEAR))) (contracts/EglContract.sol#434-440)
                - voteWeightsSum[i] = voteWeightsSum[i].add(voteWeight) (contracts/EglContract.sol#830)
        - _internalVote(msg.sender,_gasTarget,bonusEglsDue,_lockupDuration,firstEpochStartDate.add(epochLength.mul(WEEKS_IN_YEAR))) (contracts/EglContract.sol#434-440)
                - votesTotal[i] = votesTotal[i].add(_eglAmount) (contracts/EglContract.sol#834)
```

```
Reentrancy in EglContract.reVote(uint256,uint256,uint8) (contracts/EglContract.sol#510-538):
        External calls:
        - eglToken.transferFrom(msg.sender,address(this),_eglAmount) (contracts/EglContract.sol#523)
        - tallyVotes() (contracts/EglContract.sol#526)
                - eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
        State variables written after the call(s):
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,originalReleaseDate) (contracts/EglContract.sol#530-536)
                - epochGasLimitSum = epochGasLimitSum.add(int256(block.gaslimit)) (contracts/EglContract.sol#815)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,originalReleaseDate) (contracts/EglContract.sol#530-536)
                - epochVoteCount = epochVoteCount.add(1) (contracts/EglContract.sol#816)
        - _eglAmount = _eglAmount.add(_internalWithdraw(msg.sender)) (contracts/EglContract.sol#529)
                - gasTargetSum[i] = gasTargetSum[i].sub(voteWeight.mul(gasTarget)) (contracts/EglContract.sol#877)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,originalReleaseDate) (contracts/EglContract.sol#530-536)
                - gasTargetSum[i] = gasTargetSum[i].add(_gasTarget.mul(voteWeight)) (contracts/EglContract.sol#831)
        - _eglAmount = _eglAmount.add(_internalWithdraw(msg.sender)) (contracts/EglContract.sol#529)
                - tokensInCirculation = tokensInCirculation.add(voterReward) (contracts/EglContract.sol#884)
        - _eglAmount = _eglAmount.add(_internalWithdraw(msg.sender)) (contracts/EglContract.sol#529)
                - voteWeightsSum[i] = voteWeightsSum[i].sub(voteWeight) (contracts/EglContract.sol#876)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,originalReleaseDate) (contracts/EglContract.sol#530-536)
                - voteWeightsSum[i] = voteWeightsSum[i].add(voteWeight) (contracts/EglContract.sol#830)
        - _eglAmount = _eglAmount.add(_internalWithdraw(msg.sender)) (contracts/EglContract.sol#529)
                - delete voters[_voter] (contracts/EglContract.sol#867)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,originalReleaseDate) (contracts/EglContract.sol#530-536)
                - voter.voteEpoch = currentEpoch (contracts/EglContract.sol#821)
                - voter.lockupDuration = _lockupDuration (contracts/EglContract.sol#822)
                - voter.releaseDate = updatedReleaseDate (contracts/EglContract.sol#823)
                - voter.tokensLocked = _eglAmount (contracts/EglContract.sol#824)
                - voter.gasTarget = _gasTarget (contracts/EglContract.sol#825)
        - _eglAmount = _eglAmount.add(_internalWithdraw(msg.sender)) (contracts/EglContract.sol#529)
                - votesTotal[i] = votesTotal[i].sub(originalEglAmount) (contracts/EglContract.sol#881)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,originalReleaseDate) (contracts/EglContract.sol#530-536)
                - votesTotal[i] = votesTotal[i].add(_eglAmount) (contracts/EglContract.sol#834)
Reentrancy in EglContract.tallyVotes() (contracts/EglContract.sol#656-750):
        External calls:
        - _issueCreatorRewards(currentEpoch) (contracts/EglContract.sol#734)
                - eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
        State variables written after the call(s):
        - currentEpoch += 1 (contracts/EglContract.sol#736)
        - currentEpochStartDate = currentEpochStartDate.add(epochLength) (contracts/EglContract.sol#737)
Reentrancy in EglContract.vote(uint256,uint256,uint8) (contracts/EglContract.sol#478-500):
        External calls:
        - tallyVotes() (contracts/EglContract.sol#490)
                - eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
        - eglToken.transferFrom(msg.sender,address(this),_eglAmount) (contracts/EglContract.sol#492)
        State variables written after the call(s):
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,0) (contracts/EglContract.sol#493-499)
                - epochGasLimitSum = epochGasLimitSum.add(int256(block.gaslimit)) (contracts/EglContract.sol#815)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,0) (contracts/EglContract.sol#493-499)
                - epochVoteCount = epochVoteCount.add(1) (contracts/EglContract.sol#816)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,0) (contracts/EglContract.sol#493-499)
                - gasTargetSum[i] = gasTargetSum[i].add(_gasTarget.mul(voteWeight)) (contracts/EglContract.sol#831)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,0) (contracts/EglContract.sol#493-499)
                - voteWeightsSum[i] = voteWeightsSum[i].add(voteWeight) (contracts/EglContract.sol#830)
        - _internalVote(msg.sender,_gasTarget,_eglAmount,_lockupDuration,0) (contracts/EglContract.sol#493-499)
                - votesTotal[i] = votesTotal[i].add(_eglAmount) (contracts/EglContract.sol#834)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

```
INFO:Detectors:
EglContract._calculateBlockReward(int256,int256,int256).proximityRewardPercent (contracts/EglContract.sol#949) is a local variable never initialized
EglContract._calculateBlockReward(int256,int256,int256).totalRewardPercent (contracts/EglContract.sol#948) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
EglContract.vote(uint256,uint256,uint8) (contracts/EglContract.sol#478-500) ignores return value by eglToken.transferFrom(msg.sender,address(this),_eglAmount) (contracts/EglContract.sol#492)
EglContract.reVote(uint256,uint256,uint8) (contracts/EglContract.sol#510-538) ignores return value by eglToken.transferFrom(msg.sender,address(this),_eglAmount) (contracts/EglContract.sol#523)
EglContract.withdraw() (contracts/EglContract.sol#543-547) ignores return value by eglToken.transfer(msg.sender,_internalWithdraw(msg.sender)) (contracts/EglContract.sol#546)
EglContract.sweepPoolRewards() (contracts/EglContract.sol#553-572) ignores return value by eglToken.transfer(block.coinbase,Math.umin(eglToken.balanceOf(address(this)),blockReward)) (contracts/EglContract.sol#561)
EglContract.withdrawPoolTokens() (contracts/EglContract.sol#577-636) ignores return value by balancerPoolToken.transfer(msg.sender,Math.umin(balancerPoolToken.balanceOf(address(this)),poolTokensDue)) (contracts/EglContract.sol#632-635)
EglContract._issueCreatorRewards(uint256) (contracts/EglContract.sol#906-930) ignores return value by eglToken.transfer(creatorRewardsAddress,creatorRewardForEpoch) (contracts/EglContract.sol#916)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

4.2 AUTOMATED SECURITY SCAN

MYTHX:

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

Results:

Report for contracts/EglContract.sol
https://dashboard.mythx.io/#/console/analyses/b5337dd4-706a-4d9d-9024-65a72c4dd976

Line	SWC Title	Severity	Short Description
89	(SWC-131) Presence of unused variables	Low	Unused state variable "supporterEglsTotal".
90	(SWC-131) Presence of unused variables	Low	Unused state variable "poolTokensHeld".
284	(SWC-000) Unknown	Medium	Function could be marked as external.
554	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
555	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
782	(SWC-000) Unknown	Medium	Function could be marked as external.
875	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
980	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1111	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.

All relevant findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN