// HALBORN

Seascape - Lighthouse

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 18th, 2021 - October 27th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/18/2021	Ferran Celades
0.2	Document Edits	10/25/2021	Ferran Celades
0.3	Final Draft	10/27/2021	Ferran Celades
0.4	Final Draft Review	10/27/2021	Gabi Urrutia
1.0	Remediation Plan	11/01/2021	Ferran Celades
1.1	Remediation Plan Review	11/03/2021	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
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Seascape engaged Halborn to conduct a security audit on their Lighthouse smart contracts beginning on October 18th, 2021 and ending on October 27th, 2021. The security assessment was scoped to the smart contracts provided in the Github repository blocklords/lighthouse-smartcontracts

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

In summary, Halborn identified some security risks that were addressed by the Seascape team.

1.3 TEST APPROACH & METHODOLOGY

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

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

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

RISK SCALE - IMPACT

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

1 - May cause minimal or un-noticeable impact.

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

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

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following lighthouse-smartcontracts
- LighthouseAuction.sol
- LighthouseMint.sol
- LighthouseAllocation.sol
- LighthouseProject.sol
- LighthouseRegistration.sol
- LighthouseBurn.sol
- LighthouseNft.sol
- LighthousePrefund.sol
- LighthouseTierWrapper.sol
- LighthouseTier.sol
- LighthouseStake.sol
- All contracts inherited by these contracts

Commit ID: 730c6857f4c0d95aca7e9655dbd4941406cf77b7
Multiple commits where used, redacted on each issue. The final commit is the following:
Fixed Commit ID: 9f61c69568addc582a85a1531b4ba9d422e9a400

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	0	2	4

LIKELIHOOD

IMPACT

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

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - CLAIM CAN SKIP 0 LEVEL	Critical	SOLVED - 10/27/2021
HAL02 - CONTRACT LOCKED WHEN TRANSFERRING PRE-FUND	High	SOLVED - 10/27/2021
HAL03 - INVALID FEE CHECK	Low	SOLVED - 10/29/2021
HAL04 - MISSING PARAMETER CHECK	Low	SOLVED - 10/29/2021
HAL05 - TAUTOLOGY EXPRESSION	Informational	SOLVED - 10/29/2021
HAL06 - UNUSED VARIABLES	Informational	SOLVED - 10/29/2021
HAL07 - INCONSISTENT CONSTRUCTORS	Informational	SOLVED - 10/29/2021
HAL08 - FUNCTION AND REQUIRE TYPOS	Informational	SOLVED - 10/29/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) CLAIM CAN SKIP LEVEL ZERO - CRITICAL

Description:

Claim on LighthouseTier can skip claim for level 0 if the given level signature containing a level 1 is valid.

```
>>> tier.getTierLevel(accounts[0])
-1
>>> tier.tiers(accounts[0])
(0, False, 0)
>>> tier.claim(1, sig['v'], sig['r'], sig['s'])
Transaction sent: 0xb65f75751befea5da7c44e5ba0d1d0f1c83c468080388c1a5e4de415834f25b0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 201
  LighthouseTier.claim confirmed - Block: 202   Gas used: 129799 (1.08%)

<Transaction '0xb65f75751befea5da7c44e5ba0d1d0f1c83c468080388c1a5e4de415834f25b0'>
>>> tier.getTierLevel(accounts[0])
1
>>> tier.tiers(accounts[0])
(1, True, 1)
>>>
```

Code Location:

Listing 1: contracts/LighthouseTier (Lines 121,124)

```
116 function claim(uint8 level, uint8 v, bytes32 r, bytes32 s)
        external {
117     require(level >= 0 && level < 4,        "LighthouseTier:
            INVALID_PARAMETER");
118     Tier storage tier = tiers[msg.sender];
119
120     // You can't Skip tiers.
121     if (level != 0) {
122         require(tier.level + 1 == level,                "
                LighthouseTier: INVALID_LEVEL");
123     } else {
124         require(tier.usable == false,                   "
                LighhouseTier: 0_CLAIMED");
125     }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendations:

It is recommended to check if the usability is set (`tier.usable`) when the `level` is different than `0`. This will prevent the `tier.level + 1 == level` check to pass when level zero has not yet been claimed.

Remediation Plan:

SOLVED: Since the contract was already deployed on the mainnet, new wrapper contract was written. This new contract, named `LighthouseTierWrapper` solved the issue. `Seascape team` fixed the issue in commit `ef9a08c2c5f9b4a2e2c92f562daf1000c9f99199`.

3.2 (HAL-02) CONTRACT LOCKED WHEN TRANSFERRING PRE-FUND - HIGH

Description:

Calling transferPrefund on LighthouseProject to a project id whose prefund stage is not finished would set the internal transferredPrefund field to true, preventing the function to be called again. Furthermore, this method can be called with a none existing project id, locking the posibility to call transferPrefund for a future created project.

```
>>> project.totalProjects()
1
>>> project.auctions(10)
(0, 0, 0, 0, 0, False)
>>> project.transferPrefund(10)
Transaction sent: 0x97ab1dcbd8cd7333a21be1ce7702459a045b32e716944323d1a0e36932375f61
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  LighthouseProject.transferPrefund confirmed - Block: 16   Gas used: 49649 (0.41%)

<Transaction '0x97ab1dcbd8cd7333a21be1ce7702459a045b32e716944323d1a0e36932375f61'>
>>> project.auctions(10)
(0, 0, 0, 0, 0, True)
>>>
```

Code Location:

Listing 2: contracts/LighthouseProject (Lines 230,257)

```
229 function transferPrefund(uint256 id) external onlyOwner {
230     if (auctions[id].transferredPrefund) {
231         return;
232     }
233
234     uint256 cap;
235     uint256 amount;
236     (cap, amount) = prefundTotalPool(id);
237
238     if (amount < cap) {
239         // We apply SCALER multiplayer, if the cap is less than
                100
240         // It could happen if investing goes in NATIVE token.
241         uint256 scaledPercent = (cap - amount) * SCALER / (cap *
```

```
                SCALER / 100);
242
243             // allocation = 10 * SCALER / 100 * SCALED percent;
244             uint256 scaledTransferAmount = (prefunds[id].
                    scaledAllocation * scaledPercent / 100) / SCALER;
245
246             auctions[id].scaledAllocation = auctions[id].
                    scaledAllocation + scaledTransferAmount;
247             prefunds[id].scaledAllocation = prefunds[id].
                    scaledAllocation - scaledTransferAmount;
248
249             uint256 scaledCompensationAmount = (prefunds[id].
                    scaledCompensation * scaledPercent / 100) / SCALER;
250
251             auctions[id].scaledCompensation = auctions[id].
                    scaledCompensation + scaledCompensationAmount;
252             prefunds[id].scaledCompensation = prefunds[id].
                    scaledCompensation - scaledCompensationAmount;
253
254             emit TransferPrefund(id, scaledTransferAmount,
                    scaledCompensationAmount);
255         }
256
257         auctions[id].transferredPrefund = true;
258     }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendations:

The function should verify that the project exists and that its funding phase has already ended.

Remediation Plan:

SOLVED: The issue was solved in commits 73ce68b0b8405d0cf1a602d247e57fbc3a2d0b4a and f2f7391d77640b1012d34eddb323dd74277c4662.

3.3 (HAL-03) INVALID FEE CHECK - LOW

Description:

The setFees function on the LighthouseTier contract does check for the fees being gibber than zero. However, the fees should be checked against the MIN_SPEND = 10 ** 6. Otherwise, the spendFrom on the claim function will always fail if the fee is less than the forementioned value.

Code Location:

Listing 3: contracts/LighthouseTier (Lines 67,68,69,70)

```
66 function setFees(uint256[4] memory _fees) public onlyOwner {
67     require(_fees[0] > 0, "LighthouseTier: ZERO_FEE_0");
68     require(_fees[1] > 0, "LighthouseTier: ZERO_FEE_1");
69     require(_fees[2] > 0, "LighthouseTier: ZERO_FEE_2");
70     require(_fees[3] > 0, "LighthouseTier: ZERO_FEE_3");
71
72     fees[0] = _fees[0];
73     fees[1] = _fees[1];
74     fees[2] = _fees[2];
75     fees[3] = _fees[3];
76
77     emit Fees(_fees[0], _fees[1], _fees[2], _fees[3]);
78 }
```

Listing 4: contracts/LighthouseTier (Lines 139)

```
138 // Charging fee
139 require(crowns.spendFrom(msg.sender, fees[level]),  "
        LighthouseTier: CWS_UNSPEND");
```

Listing 5: contracts/crowns-token/CrownsToken.sol (Lines 513)

```
512 function spendFrom(address sender, uint256 amount) public returns(
        bool) {
```

```
513     require(amount > MIN_SPEND, "Crowns: trying to spend less than
            expected");
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

It is recommended to compare the variables with their minimum and / or maximum value when setting them if they differ from zero. Otherwise, unexpected behaviors could arise.

Remediation Plan:

SOLVED: Fixed in commit 96fa5c7cbd6970310a594a1af917f4c8b654e82f

3.4 (HAL-04) MISSING PARAMETER CHECK - LOW

Description:

The setFundCollector function on the LighthousePrefund contract does check for fundCollector being different than owner but the constructor of the same contract does not perform the same check

Code Location:

Listing 6: contracts/LighthousePrefund (Lines 38,44)

```
29 constructor(address _tier, address _submission, address _project,
       address payable _fundCollector, uint256 _chainID) {
30     require(_tier != address(0) && _submission != address(0) &&
           _project != address(0) && _fundCollector != address(0), "
           Lighthouse: ZERO_ADDRESS");
31     require(_tier != _submission, "Lighthouse: SAME_ADDRESS");
32     require(_tier != _project, "Lighthouse: SAME_ADDRESS");
33     require(_chainID > 0, "Lighthouse: ZERO_VALUE");
34
35     lighthouseTier = LighthouseTier(_tier);
36     lighthouseRegistration = LighthouseRegistration(_submission);
37     lighthouseProject = LighthouseProject(_project);
38     fundCollector = _fundCollector;
39     chainID = _chainID;
40 }
41
42 function setFundCollector(address payable _fundCollector) external
       onlyOwner {
43     require(_fundCollector != address(0), "Lighthouse:
           ZERO_ADDRESS");
44     require(_fundCollector != owner(), "Lighthouse: USED_OWNER");
45
46     fundCollector = _fundCollector;
47 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

The code should be consistent and the same checks should be applied on the setters and declaration declaration. The owner check should also be applied on the constructor.

Remediation Plan:

SOLVED: Fixed in commit 7fcd3b7cfaae3e134c3dd1ca08700c1788002c62

3.5 (HAL-05) TAUTOLOGY EXPRESSION - INFORMATIONAL

Description:

On LighthouseTier, on the claim and use function the level will always be >= than 0 since the datatype uint8 is used. This means that the level >= 0 check is redundant and only the level < 4 is required.

Code Location:

Listing 7: contracts/LighthouseTier (Lines 117)

```
116 function claim(uint8 level, uint8 v, bytes32 r, bytes32 s)
        external {
117     require(level >= 0 && level < 4,        "LighthouseTier:
            INVALID_PARAMETER");
```

Listing 8: contracts/LighthouseTier (Lines 153)

```
152 function use(address investor, uint8 level) external {
153     require(level >= 0 && level < 4,    "LighthouseTier:
            INVALID_PARAMETER");
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Removing tautology expressions would reduce computation gas costs.

Remediation Plan:

SOLVED: Fixed in commit 0c4e98aff297ce2beeba3a94eb9ab86299e6efeb

3.6 (HAL-06) UNUSED VARIABLES - INFORMATIONAL

Description:

The scaledRatio variable of the Prefund struct on the LighthouseProject contract is only set but not used on any aspect of the code. If this variable was used on a frontend, the same value could be obtained by using scaledAllocation and scaledCompensation (prefund.scaledAllocation / prefund.scaledCompensation). Removing the variable would reduce gas costs and storage costs on the smart contract.

Code Location:

Listing 9: contracts/LighthouseProject (Lines 39)

```
29 struct Prefund {
30     uint256 startTime;
31     uint256 endTime;
32     uint256[3] investAmounts;       // Amount of tokens that user
           can invest, depending on his tier
33     uint256[3] collectedAmounts;    // Amount of tokens that users
            invested so far.
34     uint256[3] pools;               // Amount of tokens that could
            be invested in the pool.
35     address token;                  // Token to accept from
           investor
36
37     uint256 scaledAllocation;       // prefund PCC allocation
38     uint256 scaledCompensation;     // prefund Crowns compensation
39     uint256 scaledRatio;            // Pool to compensation ratio
40 }
```

Listing 10: contracts/LighthouseProject (Lines 190)

```
188     auction.scaledAllocation    = auctionAllocation * SCALER;
189     auction.scaledCompensation  = auctionCompensation * SCALER;
190     prefund.scaledRatio         = prefund.scaledAllocation /
           prefund.scaledCompensation;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Unused variables should be removed, this would reduce gas storage and computation gas costs.

Remediation Plan:

SOLVED: Fixed in commit bcca74d12ef00c6236c78eda5f62f2d66095096d

3.7 (HAL-07) INCONSISTENT CONSTRUCTORS - INFORMATIONAL

Description:

The `LighthouseMint` and `LighthouseBurn` constructors do have different constructor parameters order. If the contract deployment is automated and the scripts are reused this could lead to bad behaviours.

Code Location:

Listing 11: contracts/LighthouseMint (Lines 36)

```
36 constructor(address _lighthouseAuction, address _lighthousePrefund
       , address _lighthouseTier, address _project, address _crowns) {
```

Listing 12: contracts/LighthouseBurn (Lines 36)

```
36 constructor(address _lighthouseAuction, address _lighthousePrefund
       , address _lighthouseTier, address _crowns, address _project) {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is a good practice to keep the same order of parameters to prevent copy-pasting/automation issues during deployment. The `_crowns` and `_project` parameters should match those of the other constructor.

Remediation Plan:

SOLVED: Fixed in commit `025f8bf9c552062896da7749d0ebd1372941ef6b`

3.8 (HAL-08) FUNCTION AND REQUIRE TYPOS - INFORMATIONAL

Description:

The deleteEditorEditor on the LighthouseProject contract should be named deleteEditor to keep consistency with the other declared methods.

The prefund method on the LighthousePrefund contract, does contain the require statement with the Lighthouse: REGISTRATION_NOT_INITIALIZED message, this message should be changed to Lighthouse: PREFUND_NOT_INITIALIZED to reflect the actual reason of revert.

Code Location:

Listing 13: contracts/LighthouseProject (Lines 105)

```
104 /// @notice Remove the tier user.
105 function deleteEditorEditor(address _user) external onlyOwner {
106     require(_user != address(0),                "Lighthouse:
            ZERO_ADDRESS");
107     require(editors[_user],                     "Lighthouse:
            NO_USER");
108
109     editors[_user] = false;
110
111     emit ProjectEditor(_user, false);
112 }
```

Listing 14: contracts/LighthousePrefund (Lines 55)

```
53 /// @dev v, r, s are used to ensure on server side that user
       passed KYC
54 function prefund(uint256 projectId, int8 certainTier, uint8 v,
       bytes32 r, bytes32 s) external payable {
55     require(lighthouseProject.prefundInitialized(projectId), "
           Lighthouse: REGISTRATION_NOT_INITIALIZED");
56     require(!prefunded(projectId, msg.sender), "Lighthouse:
           ALREADY_PREFUNDED");
```

```
57      require(certainTier > 0 && certainTier < 4, "Lighthouse:
            INVALID_CERTAIN_TIER");
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to fix typos in smart contracts so that the code is clearer and more understandable.

Remediation Plan:

SOLVED: Fixed in commit 7fcd3b7cfaae3e134c3dd1ca08700c1788002c62

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

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

Slither results:

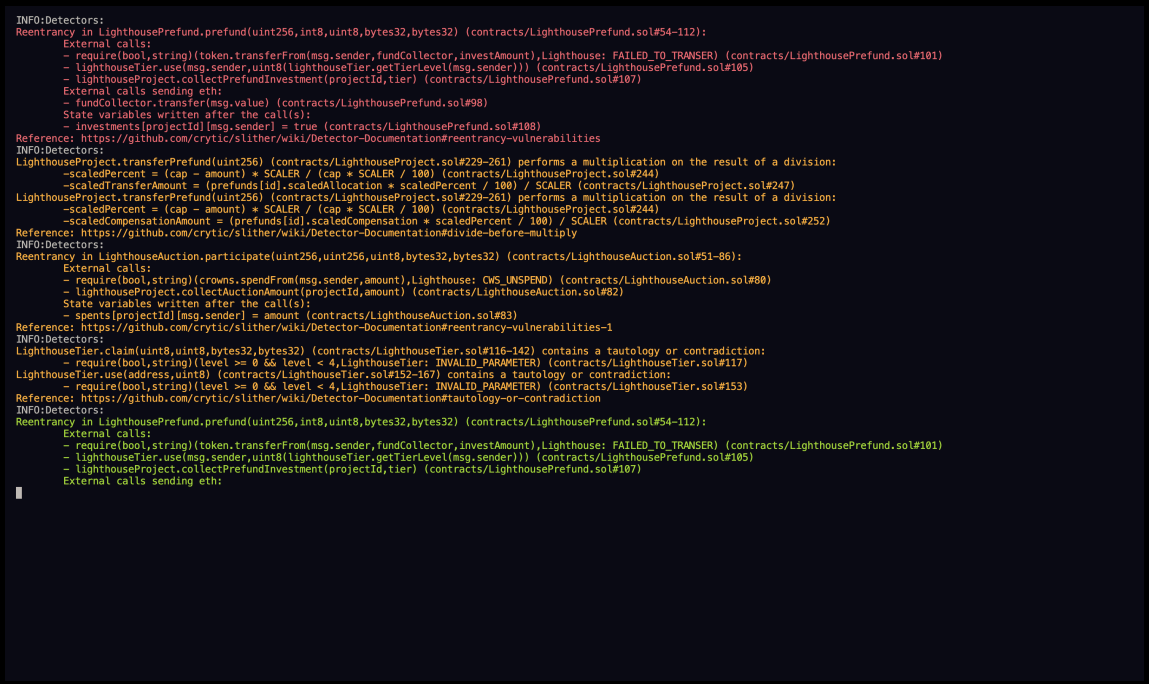

Figure 1: contracts/LighthouseAuction.sol

All reentrancy issues were false positive. Furthermore, all the vulnerabilities found were already described on the report

```
INFO:Detectors:
Reentrancy in LighthousePrefund.prefund(uint256,int8,uint8,bytes32,bytes32) (contracts/LighthousePrefund.sol#54-112):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,fundCollector,investAmount),Lighthouse: FAILED_TO_TRANSER) (contracts/LighthousePrefund.sol#101)
        - lighthouseTier.use(msg.sender,uint8(lighthouseTier.getTierLevel(msg.sender))) (contracts/LighthousePrefund.sol#105)
        - lighthouseProject.collectPrefundInvestment(projectId,tier) (contracts/LighthousePrefund.sol#107)
        External calls sending eth:
        - fundCollector.transfer(msg.value) (contracts/LighthousePrefund.sol#98)
        State variables written after the call(s):
        - investments[projectId][msg.sender] = true (contracts/LighthousePrefund.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledTransferAmount = (prefunds[id].scaledAllocation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#247)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledCompensationAmount = (prefunds[id].scaledCompensation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in LighthouseMint.mint(uint256) (contracts/LighthouseMint.sol#58-115):
        External calls:
        - require(bool,string)(lighthouseNft.mint(projectId,nftId,msg.sender,allocation,compensation,tierLevel,mintType),LighthouseMint: FAILED) (contracts/LighthouseMint.sol#109)
        State variables written after the call(s):
        - mintedNfts[projectId][msg.sender] = nftId (contracts/LighthouseMint.sol#112)
Reentrancy in LighthouseAuction.participate(uint256,uint256,uint8,bytes32,bytes32) (contracts/LighthouseAuction.sol#51-86):
        External calls:
        - require(bool,string)(crowns.spendFrom(msg.sender,amount),Lighthouse: CWS_UNSPEND) (contracts/LighthouseAuction.sol#80)
        - lighthouseProject.collectAuctionAmount(projectId,amount) (contracts/LighthouseAuction.sol#82)
        State variables written after the call(s):
        - spents[projectId][msg.sender] = amount (contracts/LighthouseAuction.sol#83)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#117)
LighthouseTier.use(address,uint8) (contracts/LighthouseTier.sol#152-167) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) ignores return value by IERC721Receiver(to).onERC721Receiv
ed(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#376-386)
```

Figure 2: contracts/LighthouseMint.sol

```
INFO:Detectors:
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledTransferAmount = (prefunds[id].scaledAllocation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#247)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledCompensationAmount = (prefunds[id].scaledCompensation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
LighthouseProject.initRegistration(uint256,uint256) (contracts/LighthouseProject.sol#117-128) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < startTime,Lighthouse: TIME_PASSED) (contracts/LighthouseProject.sol#118)
LighthouseProject.initPrefund(uint256,uint256,uint256,uint256[3],uint256[3],address) (contracts/LighthouseProject.sol#131-151) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < startTime,Lighthouse: INVALID_START_TIME) (contracts/LighthouseProject.sol#133)
LighthouseProject.initAuction(uint256,uint256,uint256) (contracts/LighthouseProject.sol#154-170) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < startTime,Lighthouse: INVALID_START_TIME) (contracts/LighthouseProject.sol#156)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(prefunds[id].endTime < block.timestamp,Lighthouse: PREFUND_PHASE) (contracts/LighthouseProject.sol#235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-22) is never used and should be removed
Counters.decrement(Counters.Counter) (node_modules/@openzeppelin/contracts/utils/Counters.sol#31-37) is never used and should be removed
Counters.reset(Counters.Counter) (node_modules/@openzeppelin/contracts/utils/Counters.sol#39-41) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (contracts/LighthouseProject.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter LighthouseProject.addEditor(address)._user (contracts/LighthouseProject.sol#95) is not in mixedCase
Parameter LighthouseProject.deleteEditorEditor(address)._user (contracts/LighthouseProject.sol#105) is not in mixedCase
Parameter LighthouseProject.initPrefund(uint256,uint256,uint256,uint256[3],uint256[3],address)._token (contracts/LighthouseProject.sol#131) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
```

Figure 3: contracts/LighthouseProject.sol

```
INFO:Detectors:
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledTransferAmount = (prefunds[id].scaledAllocation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#247)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledCompensationAmount = (prefunds[id].scaledCompensation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#117)
LighthouseTier.use(address,uint8) (contracts/LighthouseTier.sol#152-167) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
Reentrancy in LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142):
        External calls:
        - require(bool,string)(crowns.spendFrom(msg.sender,fees[level]),LighthouseTier: CWS_UNSPEND) (contracts/LighthouseTier.sol#139)
        Event emitted after the call(s):
        - Claim(msg.sender,level) (contracts/LighthouseTier.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
LighthouseProject.initRegistration(uint256,uint256) (contracts/LighthouseProject.sol#117-128) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < startTime,Lighthouse: TIME_PASSED) (contracts/LighthouseProject.sol#118)
LighthouseProject.initPrefund(uint256,uint256,uint256,uint256[3],uint256[3],address) (contracts/LighthouseProject.sol#131-151) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < startTime,Lighthouse: INVALID_START_TIME) (contracts/LighthouseProject.sol#133)
LighthouseProject.initAuction(uint256,uint256,uint256) (contracts/LighthouseProject.sol#154-170) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < startTime,Lighthouse: INVALID_START_TIME) (contracts/LighthouseProject.sol#156)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(prefunds[id].endTime < block.timestamp,Lighthouse: PREFUND_PHASE) (contracts/LighthouseProject.sol#235)
LighthouseRegistration.register(uint256) (contracts/LighthouseRegistration.sol#45-65) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= startTime,Lighthouse: NOT_STARTED_YET) (contracts/LighthouseRegistration.sol#53)
        - require(bool,string)(block.timestamp <= endTime,Lighthouse: FINISHED) (contracts/LighthouseRegistration.sol#54)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
```

Figure 4: contracts/LighthouseRegistration.sol

```
INFO:Detectors:
Reentrancy in LighthousePrefund.prefund(uint256,int8,uint8,bytes32,bytes32) (contracts/LighthousePrefund.sol#54-112):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,fundCollector,investAmount),Lighthouse: FAILED_TO_TRANSER) (contracts/LighthousePrefund.sol#101)
        - lighthouseTier.use(msg.sender,uint8(lighthouseTier.getTierLevel(msg.sender))) (contracts/LighthousePrefund.sol#105)
        - lighthouseProject.collectPrefundInvestment(projectId,tier) (contracts/LighthousePrefund.sol#107)
        External calls sending eth:
        - fundCollector.transfer(msg.value) (contracts/LighthousePrefund.sol#98)
        State variables written after the call(s):
        - investments[projectId][msg.sender] = true (contracts/LighthousePrefund.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledTransferAmount = (prefunds[id].scaledAllocation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#247)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledCompensationAmount = (prefunds[id].scaledCompensation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in LighthouseAuction.participate(uint256,uint256,uint8,bytes32,bytes32) (contracts/LighthouseAuction.sol#51-86):
        External calls:
        - require(bool,string)(crowns.spendFrom(msg.sender,amount),Lighthouse: CWS_UNSPEND) (contracts/LighthouseAuction.sol#80)
        - lighthouseProject.collectAuctionAmount(projectId,amount) (contracts/LighthouseAuction.sol#82)
        State variables written after the call(s):
        - spents[projectId][msg.sender] = amount (contracts/LighthouseAuction.sol#83)
Reentrancy in LighthouseBurn.transferMaxReserve(address) (contracts/LighthouseBurn.sol#79-88):
        External calls:
        - require(bool,string)(pcc.transferFrom(address(this),staker,stakeReserves[pccAddress]),LighthouseBurn: FAILED_TO_TRANSFER) (contracts/LighthouseBurn.sol#85)
        State variables written after the call(s):
        - stakeReserves[pccAddress] = 0 (contracts/LighthouseBurn.sol#87)
Reentrancy in LighthouseBurn.transferStakeReserve(address,uint256) (contracts/LighthouseBurn.sol#66-77):
        External calls:
        - require(bool,string)(pcc.transferFrom(address(this),staker,amount),Lighthouse: FAILED_TO_TRANSFER) (contracts/LighthouseBurn.sol#74)
        State variables written after the call(s):
        - stakeReserves[pccAddress] = stakeReserves[pccAddress] - amount (contracts/LighthouseBurn.sol#76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#117)
```

Figure 5: contracts/LighthouseBurn.sol

```
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) ignores return value by IERC721Receiver(to).onERC721Receiv
ed(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#376-386)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
LighthouseNft.setOwner(address)._owner (contracts/LighthouseNft.sol#86) shadows:
        - Ownable._owner (node_modules/@openzeppelin/contracts/access/Ownable.sol#20) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#376)' in ERC721._checkOnERC721Received(address,add
ress,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) potentially used before declaration: retval == IERC721Receiver.onERC721Received.selector (node_modu
les/@openzeppelin/contracts/token/ERC721/ERC721.sol#377)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#378)' in ERC721._checkOnERC721Received(address,add
ress,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) potentially used before declaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token
/ERC721/ERC721.sol#379)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#378)' in ERC721._checkOnERC721Received(address,add
ress,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (n
ode_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#383)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Reentrancy in LighthouseNft.mint(uint256,uint256,address,uint256,uint256,int8,uint8) (contracts/LighthouseNft.sol#60-74):
        External calls:
        - _safeMint(_to,_tokenId) (contracts/LighthouseNft.sol#65)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#376-386)
        State variables written after the call(s):
        - paramsOf[_tokenId] = Params(_allocation,_compensation,_tier,_type) (contracts/LighthouseNft.sol#67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in LighthouseNft.mint(uint256,uint256,address,uint256,uint256,int8,uint8) (contracts/LighthouseNft.sol#60-74):
        External calls:
        - _safeMint(_to,_tokenId) (contracts/LighthouseNft.sol#65)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#376-386)
        Event emitted after the call(s):
        - Minted(_to,_tokenId,_allocation,_compensation,_tier,_type,projectId) (contracts/LighthouseNft.sol#71)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#382-384)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-36) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#32-34)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#195-215) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#207-210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - 0.8.0 (contracts/LighthouseNft.sol#3)
```

Figure 6: contracts/LighthouseNft.sol

```
INFO:Detectors:
Reentrancy in LighthousePrefund.prefund(uint256,int8,uint8,bytes32,bytes32) (contracts/LighthousePrefund.sol#54-112):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,fundCollector,investAmount),Lighthouse: FAILED_TO_TRANSER) (contracts/LighthousePrefund.sol#101)
        - lighthouseTier.use(msg.sender,uint8(lighthouseTier.getTierLevel(msg.sender))) (contracts/LighthousePrefund.sol#105)
        - lighthouseProject.collectPrefundInvestment(projectId,tier) (contracts/LighthousePrefund.sol#107)
        External calls sending eth:
        - fundCollector.transfer(msg.value) (contracts/LighthousePrefund.sol#98)
        State variables written after the call(s):
        - investments[projectId][msg.sender] = true (contracts/LighthousePrefund.sol#108)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledTransferAmount = (prefunds[id].scaledAllocation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#247)
LighthouseProject.transferPrefund(uint256) (contracts/LighthouseProject.sol#229-261) performs a multiplication on the result of a division:
        -scaledPercent = (cap - amount) * SCALER / (cap * SCALER / 100) (contracts/LighthouseProject.sol#244)
        -scaledCompensationAmount = (prefunds[id].scaledCompensation * scaledPercent / 100) / SCALER (contracts/LighthouseProject.sol#252)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#117)
LighthouseTier.use(address,uint8) (contracts/LighthouseTier.sol#152-167) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
Reentrancy in LighthousePrefund.prefund(uint256,int8,uint8,bytes32,bytes32) (contracts/LighthousePrefund.sol#54-112):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,fundCollector,investAmount),Lighthouse: FAILED_TO_TRANSER) (contracts/LighthousePrefund.sol#101)
        - lighthouseTier.use(msg.sender,uint8(lighthouseTier.getTierLevel(msg.sender))) (contracts/LighthousePrefund.sol#105)
        - lighthouseProject.collectPrefundInvestment(projectId,tier) (contracts/LighthousePrefund.sol#107)
        External calls sending eth:
        - fundCollector.transfer(msg.value) (contracts/LighthousePrefund.sol#98)
        State variables written after the call(s):
        - tiers[projectId][msg.sender] = tier (contracts/LighthousePrefund.sol#109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142):
        External calls:
        - require(bool,string)(crowns.spendFrom(msg.sender,fees[level]),LighthouseTier: CWS_UNSPEND) (contracts/LighthouseTier.sol#139)
```

Figure 7: contracts/LighthousePrefund.sol

```
INFO:Detectors:
LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#117)
LighthouseTier.use(address,uint8) (contracts/LighthouseTier.sol#152-167) contains a tautology or contradiction:
        - require(bool,string)(level >= 0 && level < 4,LighthouseTier: INVALID_PARAMETER) (contracts/LighthouseTier.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
Reentrancy in LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142):
        External calls:
        - require(bool,string)(crowns.spendFrom(msg.sender,fees[level]),LighthouseTier: CWS_UNSPEND) (contracts/LighthouseTier.sol#139)
        Event emitted after the call(s):
        - Claim(msg.sender,level) (contracts/LighthouseTier.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
LighthouseTier.claim(uint8,uint8,bytes32,bytes32) (contracts/LighthouseTier.sol#116-142) compares to a boolean constant:
        -require(bool,string)(tier.usable == false,LighhouseTier: 0_CLAIMED) (contracts/LighthouseTier.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - ^0.8.0 (contracts/LighthouseTier.sol#2)
        - 0.8.0 (contracts/crowns/CrownsInterface.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#20-22) is never used and should be removed
Counters.current(Counters.Counter) (node_modules/@openzeppelin/contracts/utils/Counters.sol#21-23) is never used and should be removed
Counters.decrement(Counters.Counter) (node_modules/@openzeppelin/contracts/utils/Counters.sol#31-37) is never used and should be removed
Counters.increment(Counters.Counter) (node_modules/@openzeppelin/contracts/utils/Counters.sol#25-29) is never used and should be removed
Counters.reset(Counters.Counter) (node_modules/@openzeppelin/contracts/utils/Counters.sol#39-41) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (contracts/LighthouseTier.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.0 (contracts/crowns/CrownsInterface.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
```

Figure 8: contracts/LighthouseTier.sol

THANK YOU FOR CHOOSING

// HALBORN