SHELLBOXES

Unicrypt's ILO V7

Smart Contract Security Audit

Prepared by ShellBoxes

Dec 24th, 2022 - Jan 9th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Unicrypt
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/chainsulting/ilov7-audit	4571798816bda734fa60edde65c047f09d691762

Re-Audit

Repository	Commit Hash
https://github.com/chainsulting/ilo7-audit	4083b43f823a3b226f03e27e5acbd40cb1d4885b

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction 4
1.1 About Unicrypt . . . 4
1.2 Approach & Methodology . . . 4
1.2.1 Risk Methodology . . . 5

2 Findings Overview 6
2.1 Disclaimer . . . 6
2.2 Summary . . . 6
2.3 Key Findings . . . 6

3 Finding Details 8
SHB.1 An Admin Can Become a Manager . . . 8
SHB.2 Admin Can Disable Presale Creation by Setting High Eth Creation Fee . . . 9
SHB.3 Potential Loss of Functionality in setFacetCuts Function . . . 10
SHB.4 Denial Of Service Vulnerability Through Owner Finalization Time Frame . . . 11
SHB.5 Centralization Risk . . . 13
SHB.6 Locked Ether . . . 19
SHB.7 WETH address can be manipulated . . . 22

4 Best Practices 24
BP.1 Merkle Tree In The Whitelist Contract . . . 24
BP.2 Remove Unnecessary Check for Address Zero in _removeAdmin Function 24
BP.3 Remove Unnecessary Initialization of totalSplitPercentage . . . 25

5 Tests 27

6 Conclusion 44

7 Scope Files 45
7.1 Audit . . . 45
7.2 Re-Audit . . . 48

8 Disclaimer 51

1 Introduction

Unicrypt engaged ShellBoxes to conduct a security assessment on the ILO V7 Presale Smart Contracts beginning on Dec 24th, 2022 and ending Jan 9th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Unicrypt

Started in June 2020, Unicrypt provides an ever-growing suite of decentralized services. The objective is to bring value to the DeFi space as a whole by delivering disruptive, flexible and audited technology.

Issuer	Unicrypt
Website	`https://unicrypt.network/`
Type	Solidity Smart Contract
Documentation	`https://docs.unicrypt.network`
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Disclaimer

This audit report highlights security issues that were identified within the scope of the audit, which includes all smart contracts in the Ilov7 repository. Despite the client's developers having performed unit tests with 100% coverage of the audited contracts, the client has not taken any action to address or mitigate the risks associated with most of the identified issues in this report. Therefore, we advise the client to take the necessary action to fix as many issues as possible in their next version of the project to ensure the security and integrity of their smart contracts.

2.2 Summary

The following is a synopsis of our conclusions from our analysis of the Unicrypt's ILO V7 implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.3 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 high-severity, 4 medium-severity, 2 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. An Admin Can Become a Manager	HIGH	Acknowledged
SHB.2. Admin Can Disable Presale Creation by Setting High Eth Creation Fee	MEDIUM	Acknowledged
SHB.3. Potential Loss of Functionality in setFacetCuts Function	MEDIUM	Acknowledged

SHB.4. Denial Of Service Vulnerability Through Owner Finalization Time Frame	MEDIUM	Acknowledged
SHB.5. Centralization Risk	MEDIUM	Acknowledged
SHB.6. Locked Ether	LOW	Acknowledged
SHB.7. WETH address can be manipulated	LOW	Acknowledged

3 Finding Details

SHB.1 An Admin Can Become a Manager

- Severity : HIGH
- Status : Acknowledged
- Likelihood : 2
- Impact : 3

Description:

The AdminRegistry contract manages the admins by adding and removing to an EnumerableSet. This contract can be implemented using the AdminRegistryImplementer .However, any admin can call the setAdminRegistry function in the AdminRegistryImplementer ,and set a contract where they are the manager since they are the deployer, and take full control over the contract.

Files Affected:

SHB.1.1: AdminRegistryImplenter.sol

```
45 function setAdminRegistry(
46         IAdminRegistry _adminRegistry
47     ) external onlyAdmin {
48         require(address(_adminRegistry).code.length > 0, "ARI:
               ↪ NO_CONTRACT");
49         adminRegistry = _adminRegistry;
50     }
```

Recommendation:

Consider implementing the setAdminRegistry with an access control allowing only the managers to upgrade the AdminRegistry contract. Additionally, it would be better if the AdminRegistry is not updated until the majority of admins approve it.

Updates

The Unicrypt team acknowledged the risk, stating that they are planning to put in place Multi-signature wallets for management activities, to mitigate the risk.

SHB.2 Admin Can Disable Presale Creation by Setting High Eth Creation Fee

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The _chargeCreationFee function is responsible for charging the eth creation fee for a new presale. This fee is set by the admin using the setFees function, which allows the admin to specify the value of the eth creation fee. However, there is no check in place to ensure that the fee is set to a reasonable amount. An attacker with admin privileges could exploit this vulnerability by setting the _ethCreationFee to a very high value, effectively making it impossible for users to create new presales and disabling this functionality for the contract.

Files Affected:

SHB.2.1: PresaleFactory.sol

```
619  function _chargeCreationFee(uint256 _feeProfile) private returns (bool)
         ↪  {
620          uint256 creationFee = presaleSettings.getEthCreationFee(
                 ↪ _feeProfile);
621          require(msg.value == creationFee, "PF: INVALID_FEE_AMOUNT");
622          (bool sentFee, ) = presaleSettings.getEthFeeReceiver().call{
623              value: creationFee
624          }("");
625          return sentFee;
```

Recommendation:

To mitigate this issue, it is recommended to add a check in the setFees function to ensure that the _ethCreationFee is set to a reasonable amount. This could be done by adding a require statement to limit the maximum value that can be set for the fee.

Updates

The Unicrypt team acknowledged the risk, stating that the likelihood of an admin causing a denial of service is low.

SHB.3 Potential Loss of Functionality in setFacetCuts Function

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The setFacetCuts function allows an admin to set the available facet cuts for a Diamond contract. However, there is currently no check to ensure that the array of facet cuts passed as an argument, _facetCuts, is not empty. If an empty array is passed, all previous facetCuts will be deleted and no new ones will be added, effectively disabling the functionality for any deployed presales that depend on the availability of facet cuts.

Files Affected:

SHB.3.1: PresaleGenerator.sol

```
73      function setFacetCuts(
74          IDiamond.FacetCut[] calldata _facetCuts
75      ) external onlyAdmin {
76          require(canSetFacets, "PG: DISABLED");
```

```
77
78          // remove all old facets
79          delete facetCuts;
80
81          // add new facets
82          for (uint256 i = 0; i < _facetCuts.length; i++) {
83              facetCuts.push(_facetCuts[i]);
84          }
85      }
```

Recommendation:

Consider adding a check to ensure that the _facetCuts array is not empty before allowing the deletion of previous facetCuts ,and the addition of new ones. This could be done by adding a require statement at the beginning of the function to check the length of the array:

SHB.3.2: PresaleGenerator.sol

```
require(_facetCuts.length > 0, "FC: NO_FACET_CUTS_PROVIDED");
```

This would prevent the loss of functionality by ensuring that at least one facet cut is provided before allowing the update to proceed.

Updates

The Unicrypt Team acknowledged this issue, stating that it is unlikely for an admin to cause a DoS.

SHB.4 Denial Of Service Vulnerability Through Owner Finalization Time Frame

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The function adminSetOwnerFinalizationFrame allows the contract administrator to set the owner finalization time frame, but does not include any checks to ensure that the specified time frame is reasonable. If the presale owner is not available and the time frame is set to a large value, this could potentially cause a denial of service for the contract, as the owner would not be able to finalize the presale within the allotted time frame.

Files Affected:

SHB.4.1: PresaleRestictedFacet.sol

```
275     function adminSetOwnerFinalizationFrame(uint64 _time) external
            ↪ onlyAdmin {
276         LibPresaleInfo.setOwnerFinalization(_time);
277     }
```

SHB.4.2: PresaleParticipantFacet.sol

```
155     function finalizePresale() external override nonReentrant {
156         LibPresaleStatus.enforceIsSuccessful();
157         LibPresaleStatus.enforceLpNotGenerated();
158
159         // get storage variables
160         IPresaleSettings settings = IPresaleSettings(LibPresaleInfo.
                ↪ settings());
161         LibPresaleInfo.Numbers memory numbers = LibPresaleInfo.numbers();
162
163         // check for owner exclusive finalization timeframe
164         require(
165             numbers.endTime + numbers.ownerFinalizationFrame <
166                 block.timestamp
167                 msg.sender == LibDiamond.contractOwner(),
168             "PPF: ONLY_OWNER_TIMEFRAME"
169         );
```

Recommendation:

The function should include checks to ensure that the specified ownerFinalizationFrame is reasonable and does not exceed a certain threshold (e.g. a few days or weeks). This can help prevent a denial of service attack by ensuring that the presale owner has a reasonable amount of time to finalize the presale.

Updates

The Unicrypt team acknowledged this issue, stating it is unlikely to happen.

SHB.5 Centralization Risk

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The PresaleSettings contract allows updating fees, tokens, and some contracts addresses. However, a single admin has too much access to update some critical state variables. Additionally, an admin can force fail any presale at any point, which is a huge centralization risk.

Exploit Scenario:

As an example :

- An admin can call the setFeeAddresses with their address in the arguments and receive all the fees.

Files Affected:

SHB.5.1: PresaleLockForwarder.sol

```
164     function replaceLockerContract(
165         address _locker,
```

```
        bool _active
    ) external onlyAdmin {
        require(_locker.code.length > 0, "PLF: NO_CONTRACT");
        address ammFactory = ILPLocker(_locker).uniswapFactory();
        require(isListedAmm(ammFactory), "PLF: AMM_NOT_LISTED");
        ammInfo[ammFactory] = AmmInfo(_locker, _active);
        emit ReplacedLocker(ammFactory, _locker, _active);
    }
```

SHB.5.2: PresaleSettings.sol

```
function setFeeAddresses(
        address payable _ethAddress,
        address payable _nonEthAddress,
        address _saleTokenFeeAddress
    ) external onlyAdmin {
        require(
            _ethAddress != address(0) &&
                _nonEthAddress != address(0) &&
                _saleTokenFeeAddress != address(0),
            "PS: Zero address"
        );
        feeSettings.ethFeeReceiver = _ethAddress;
        feeSettings.nonEthFeeReceiver = _nonEthAddress;
        feeSettings.saleTokenFeeReceiver = _saleTokenFeeAddress;
    }
```

SHB.5.3: PresaleSettings.sol

```
function setFees(
        uint16 _baseTokenFee,
        uint16 _saleTokenFee,
        uint16 _referralFee,
        uint16 _referralFeeSplit,
        uint256 _ethCreationFee
    ) external onlyAdmin {
```

```
164         require(_ethCreationFee >= DENOMINATOR, "PS: Fee too low");
165         require(
166             _baseTokenFee <= DENOMINATOR &&
167                 _saleTokenFee <= DENOMINATOR &&
168                 _referralFee <= DENOMINATOR &&
169                 _referralFeeSplit <= DENOMINATOR,
170             "PS: Fee too high"
171         );
172         feeSettings.baseTokenFee = _baseTokenFee;
173         feeSettings.saleTokenFee = _saleTokenFee;
174         feeSettings.ethCreationFee = _ethCreationFee;
175         feeSettings.referralFee = _referralFee;
176         feeSettings.referralFeeSplit = _referralFeeSplit;
177 }
```

SHB.5.4: PresaleSettings.sol

```
187 function setEmergencyFees(
188         uint16 _none,
189         uint16 _entry,
190         uint16 _mid,
191         uint16 _high
192     ) external onlyAdmin {
193         // check fee values
194         require(
195             _none <= DENOMINATOR &&
196                 _entry <= DENOMINATOR &&
197                 _mid <= DENOMINATOR &&
198                 _high <= DENOMINATOR,
199             "PS: FEE_TOO_HIGH"
200         );
201         require(_entry <= _none, "PS: ENTRY_TOO_LOW");
202         require(_mid <= _entry, "PS: MID_TOO_LOW");
203         require(_high <= _mid, "PS: HIGH_TOO_LOW");
204         // set emergency fee values
```

```
205         emergencyWithdrawlFees[IStaking.Tier.None] = _none;
206         emergencyWithdrawlFees[IStaking.Tier.Entry] = _entry;
207         emergencyWithdrawlFees[IStaking.Tier.Mid] = _mid;
208         emergencyWithdrawlFees[IStaking.Tier.High] = _high;
209     }
```

SHB.5.5: PresaleSettings.sol

```
216 function setDefaultReferrer(
217         address payable _defaultReferrer
218     ) external onlyAdmin {
219         require(_defaultReferrer != address(0), "PS: Zero address");
220         defaultReferrer = _defaultReferrer;
221     }
```

SHB.5.6: PresaleSettings.sol

```
230 function setMinLockingLiquidity(uint64 _minLiquidity) external onlyAdmin
        ↪ {
231         require(_minLiquidity <= DENOMINATOR, "PS: Liquidity too high");
232         generalSettings.minLiquidityPercentage = _minLiquidity;
233     }
```

SHB.5.7: PresaleSettings.sol

```
239 function setMinLockingDuration(uint64 _minDuration) external onlyAdmin {
240         generalSettings.minLiquidityLockingDuration = _minDuration;
241     }
```

SHB.5.8: PresaleSettings.sol

```
252 function setOwnerFinalizeDuration(uint64 _duration) external onlyAdmin {
253         require(_duration <= 3 days, "PS: Duration too long");
254         generalSettings.ownerFinalizeDuration = _duration;
255     }
```

SHB.5.9: PresaleSettings.sol

```
264 function setWhitelist(IWhitelist _whitelist) external onlyAdmin {
```

```
265         require(address(_whitelist).code.length > 0, "PS: NO_CONTRACT");
266         whitelist = _whitelist;
267     }
```

SHB.5.10: PresaleSettings.sol

```
276 function setWeth(IWETH _weth) external onlyAdmin {
277         require(address(_weth).code.length > 0, "PS: NO_CONTRACT");
278         weth = _weth;
279 }
```

SHB.5.11: PresaleSettings.sol

```
288 function setTokenVesting(ITokenVesting _tokenVesting) external onlyAdmin
        ↪ {
289         require(address(_tokenVesting).code.length > 0, "PS: NO_CONTRACT
                ↪ ");
290         tokenVesting = _tokenVesting;
291 }
```

SHB.5.12: PresaleSettings.sol

```
300     function setLockForwarder(
301         IPresaleLockForwarder _lockForwarder
302     ) external onlyAdmin {
303         require(address(_lockForwarder).code.length > 0, "PS: NO_CONTRACT
                ↪ ");
304         lockForwarder = _lockForwarder;
305     }
```

SHB.5.13: PresaleSettings.sol

```
314     function setStaking(IStaking _staking) external onlyAdmin {
315         require(address(_staking).code.length > 0, "PS: NO_CONTRACT");
316         staking = _staking;
317     }
```

SHB.5.14: PresaleSettings.sol

```
334 function addFeeProfile(
335         bool _activate,
336         uint16 _baseTokenFeeDiscount,
337         uint16 _saleTokenFeeDiscount,
338         uint16 _creationFeeDiscount,
339         uint64 _whitelistSlots
340     ) external onlyAdmin {
341         require(
342             _baseTokenFeeDiscount <= DENOMINATOR &&
343                 _saleTokenFeeDiscount <= DENOMINATOR &&
344                 _creationFeeDiscount <= DENOMINATOR,
345             "PS: Fee too high"
346         );
347         totalFeeProfiles++;
348         feeProfiles[totalFeeProfiles] = FeeProfile(
349             _activate,
350             _baseTokenFeeDiscount,
351             _saleTokenFeeDiscount,
352             _creationFeeDiscount,
353             _whitelistSlots
354         );
355     }
```

SHB.5.15: PresaleSettings.sol

```
365 function toggleFeeProfile(
366         bool _activate,
367         uint256 _index
368     ) external onlyAdmin {
369         require(
370             _index > 0 && _index <= totalFeeProfiles,
371             "PS: PROFILE_NOT_FOUND"
372         );
373         feeProfiles[_index].active = _activate;
374 }
```

SHB.5.16: PresaleRestrictedFacet.sol

```
266 function adminForceFail() external onlyAdmin {
267         LibPresaleStatus.enforceLpNotGenerated();
268         _forceFailure();
269     }
```

Recommendation:

To help ensure that changes to the PresaleSettings contract are made with the consensus of the admins, consider implementing a voting system where the majority of admins must accept the change in order for it to be applied. This can help prevent a single admin from making changes to the settings without the agreement of the rest of the group, which could potentially be harmful to the contract or its users.

Updates

The Unicrypt team acknowledged this issue for the following reason, it is unlikely for the settings to be compromised by an owner.

SHB.6 Locked Ether

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The userDeposit function allows users to deposit either a special ERC20 token (baseToken) or a native token into a presale, depending on the preference of the presale owner. However, if a user deposits both the baseToken and ETH, the same amount of ETH is not refunded to the user and remains locked inside the contract.

Files Affected:

SHB.6.1: PresaleParticipantFacet.sol

```
272 function _userDeposit(uint256 _amount) internal {
273         // check if base token is native token
274         uint256 amountIn = LibPresaleInfo.generalInfo().baseIsNative
275             ? msg.value
276             : _amount;
277 
278         // get storage variables
279         LibPresaleBuyers.BuyerInfo storage buyer = LibPresaleBuyers
280             .getBuyerInfo(msg.sender);
281         LibPresaleInfo.Numbers memory numbers = LibPresaleInfo.numbers();
282         LibPresaleInfo.GeneralInfo memory info = LibPresaleInfo.
            ↪ generalInfo();
283         LibPresaleStatus.PresaleStatus storage status = LibPresaleStatus
284             .diamondStorage();
285         bool noHardcap = info.presaleType ==
286             LibPresaleInfo.PresaleType.NO_HARDCAP;
287 
288         // get base token allowance for deposit
289         uint256 allowance = numbers.maxSpendPerBuyer - buyer.
            ↪ baseDeposited;
290         uint256 remainingBaseToken = noHardcap
291             ? type(uint256).max - status.totalBaseTokensCollected
292             : numbers.hardcap - status.totalBaseTokensCollected;
293         allowance = allowance > remainingBaseToken
294             ? remainingBaseToken
295             : allowance;
296 
297         // check if amount is greater than deposit allowance
298         if (amountIn > allowance) {
299             amountIn = allowance;
300         }
```

```
301
302         // check sale token amount
303         uint256 saleTokenAmount = noHardcap
304             ? 0
305             : _calculateSaleTokenAmount(amountIn, numbers.tokenPrice);
306
307         // update storage variables
308         if (buyer.baseDeposited == 0) {
309             status.totalBuyers++;
310         }
311         buyer.baseDeposited += amountIn;
312         buyer.saleTokensOwed += saleTokenAmount;
313         status.totalBaseTokensCollected += amountIn;
314         status.totalSaleTokensSold += saleTokenAmount;
315
316         // return unused native tokens
317         if (info.baseIsNative && amountIn < msg.value) {
318             (bool sent, ) = msg.sender.call{value: msg.value - amountIn
                ↪ }("");
319             require(sent, "PPF: REFUND_FAILED");
320         }
321
322         // send non native base tokens to Presale
323         if (!info.baseIsNative) {
324             TransferHelper.safeTransferFrom(
325                 info.baseToken,
326                 msg.sender,
327                 address(this),
328                 amountIn
329             );
330         }
```

Recommendation:

Consider requiring the msg.value to be equal to zero when the base token is a non-native token.

Updates

The Unicrypt team acknowledged the risk, stating it is unlikely to happen.

SHB.7 WETH address can be manipulated

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The WETH address in the contract is not checked for validity, which means that the contract deployer could potentially manipulate the contract by specifying a malicious WETH contract address instead of the expected contract. This could have serious consequences for the contract's users.

Files Affected:

SHB.7.1: PresaleSettings.sol

```
63      constructor(
64          address payable _defaultReferrer,
65          address _whitelist,
66          address _adminRegistry,
67          address _weth,
68          address _tokenVesting,
69          address _lockForwarder,
70          address _staking
71      ) AdminRegistryImplementer(_adminRegistry) {
```

```
72          require(_defaultReferrer != address(0), "PS: Zero address");
73          require(
74              _whitelist.code.length > 0 &&
75                  _weth.code.length > 0 &&
76                  _tokenVesting.code.length > 0 &&
77                  _lockForwarder.code.length > 0 &&
78                  _staking.code.length > 0,
79              "PS: NO_CONTRACT"
80          );
81
82          defaultReferrer = _defaultReferrer;
83
84          // set references to other contracts
85          whitelist = IWhitelist(_whitelist);
86          weth = IWETH(_weth);
```

Recommendation:

Consider initializing the WETH address as a constant in the contract declaration. This will ensure that the WETH address cannot be manipulated by the deployer ,and will make it clear to the users of the contract what the intended WETH address is.

Updates

The Unicrypt team acknowledged the risk, stating that the address is initialized in the constructor to enable deploying the contract in different chains.

4 Best Practices

BP.1 Merkle Tree In The Whitelist Contract

Description:

Using a Merkle tree to implement a whitelist could potentially have some benefits compared to other approaches, such as reducing the gas cost of querying the whitelist and allowing for efficient updates to the whitelist.

To implement a whitelist using a Merkle tree, you would first need to store the hashes of the whitelisted addresses in the tree offchain ,and store the merkle root in the contract. You could then use the isWhitelisted function to check if a given address is on the whitelist by calculating the hash of the address and verifying that it is included in the tree.

To add or remove an address from the whitelist, you would need to update the tree by inserting or deleting the hash of the address. This would require recalculating the hashes of the affected nodes in the tree,and changing the Merkle root in the contract.

Overall, using a Merkle tree for a whitelist may be a good solution if you need to efficiently check if a number of addresses are on the whitelist.

Status - Acknowledged

The Unicrypt team acknowledged this best practice, and they are planning to implement it.

BP.2 Remove Unnecessary Check for Address Zero in _removeAdmin Function

Description:

The _removeAdmin function currently contains a check to ensure that the address being removed is not equal to address zero. However, this check is unnecessary, as the _addAdmin function already enforces the requirement that new admins must not be address zero. Therefore, the check can be safely removed without affecting the functionality of the contract.

Removing this unnecessary check can help simplify the code and make it easier to understand and maintain. It can also potentially reduce the gas cost of executing the function, as the check is an extra step that is not needed.

Files Affected:

BP.2.1: AdminRegistry.sol

```
187     function _removeAdmin(address _oldAdmin) internal {
188         require(_oldAdmin != address(0), "AR: ZERO_ADDRESS");
189         require(isAdmin(_oldAdmin), "AR: NOT_REGISTERED");
```

Status - Acknowledged

The Unicrypt team acknowledged this best practice.

BP.3 Remove Unnecessary Initialization of totalSplitPercentage

Description:

The variable totalSplitPercentage is initialized to zero at the beginning of the _validateAmmParams function. However, this initialization is unnecessary, as the value of totalSplitPercentage is immediately overwritten in the following line of code. Therefore, it is recommended to remove the unnecessary initialization of totalSplitPercentage to zero. This will help simplify the code and reduce the gas cost of executing the function.

Files Affected:

BP.3.1: PresaleFactory.sol

```
485     function _validateAmmParams(
486         LibPresaleInfo.AmmParams memory _ammParams
487     ) private view {
488         require(
```

```
489             _ammParams.ammIndexes.length == _ammParams.splitPercentages.
                    ↪ length,
490             "PF: INVALID_AMM_PARAMS"
491         );
492
493         // define variables to check amm parameters for duplicates and
                ↪ total split percentage
494         uint256 totalSplitPercentage = 0;
```

Status - Acknowledged

The Unicrypt team acknowledged this best practice for the reason being their utmost care for the code readability.

5 Tests

Results:

→ AdminRegistry

✓ Should initialize contract correctly

→ Add single admin

✓ Should add single admin as non-manager by manager

✓ Should add single admin as manager by manager

✓ Should fail adding single admin by not-manager

✓ Should fail adding zero address as admin

✓ Should fail adding already registered admin again

→ Remove single admin

✓ Should remove single manager by manager

✓ Should remove single admin by manager

✓ Should fail removing single admin by not-manager

✓ Should fail removing single admin with zero address

✓ Should fail removing single not registered admin

✓ Should fail removing single last admin

✓ Should fail removing single last manager

→ Add multiple admins

✓ Should add multiple admins by manager

✓ Should add multiple managers by manager

✓ Should add multiple admins and managers by manager

✓ Should fail adding multiple admin by not-manager

✓ Should fail adding multiple admins with different array sizes

✓ Should fail adding zero address as admin

✓ Should fail adding already registered admin again

→ Remove multiple admins

✓ Should remove multiple admins by manager

✓ Should fail removing multiple admins by not-manager

✓ Should fail removing multiple admin swith zero address

✓ Should fail removing multiple not registered admins

✓ Should fail removing single last admin

→ AdminRegistryImplementer

✓ Should initialize admin registry implementer correctly

✓ Should set admin registry address by admin

✓ Should fail setting admin registry address by non-admin

✓ Should fail setting admin registry address with invalid address

✓ Should use correct admin list

→ Presale Diamond

✓ should have five facets – call to facetAddresses function

✓ should have the right function selectors – call to facetFunctionSelectors function

✓ should associate selectors to facets correctly – multiple calls to facetAddress function

✓ Should fail initializing Numbers by non factory

✓ Should revert on calling unknown function

✓ Should store facets correctly

✓ Should return correct settings contract address

✓ Should return correct factory contract address

✓ Should fail adding facet with zero address

✓ Should fail adding facet with invalid address

✓ Should fail adding facet no selector

✓ Should fail adding facet invalid action type

✓ Should return supported interfaces correctly

→ Presale OwnershipFacet

✓ Should initialize owner correctly

✓ Should transfer ownership by owner

✓ Should fail transferring ownership by non-owner

→ Restricted Presale Facet

→ Owner functions

→ Withdraw sale tokens

✓ Should withdraw sale tokens on failure

✓ Should send sale tokens back to owner on withdrawl

✓ Should fail withdrawing sale token if presale is not failed

✓ Should fail withdrawing sale token by non owner

→ Add token holding requirement

✓ Should set token holding requirement

✓ Should fail setting holding requirement with invalid token

✓ Should fail setting holding requirement with invalid amount

✓ Should fail setting holding requirement if presale is not queued

✓ Should fail setting holding requirement by non owner

→ Update presale times

✓ Should postpone presale

✓ Should fail postponing with invalid start time

✓ Should fail postponing with invalid end time

✓ Should fail postponing with invalid round1 start time

✓ Should set free emergency withdraw on postponing

✓ Should fail postponing if presale is failed

✓ Should fail postponing by non owner

→ Force failure

✓ Should force presale to fail

✓ Should fail forcing failed presale to fail

✓ Should fail forcing presale to fail by non owner

✓ Should fail forcing failed if lp was created

→ Set token vesting

✓ Should set vesting params

✓ Should fail setting vesting params with invalid percentage

✓ Should fail forcing presale to fail by non owner

✓ Should set free emergency withdraw

→ Admin functions

→ Force failure

✓ Should force presale to fail

✓ Should fail forcing failed presale to fail

✓ Should fail forcing presale to fail by non owner

✓ Should fail forcing failed if lp was created

→ Owner finalization frame

✓ Should set owner finalization frame

✓ Should fail setting owner finalization frame by non owner

→ Custom whitelist

✓ Should set custom whitelist

✓ Should fail setting custom whitelist by non owner

✓ Should fail setting custom whitelist with invalid contract address

✓ Should fail setting custom whitelist for non discounted presale

→ PresaleFactory

✓ Should initialize factory correctly

✓ Should fail creating factory with invalid addresses

→ Calculate sale token amount required

✓ Should calculate scenario 0

✓ Should calculate scenario 1

✓ Should calculate scenario 2

✓ Should calculate scenario 3

✓ Should calculate scenario 4

- ✓ Should fail calculating amount required with invalid liquidity percent
- ✓ Should fail calculating amount required with invalid sale token fee

→ Create presale

→ Regular presale

- ✓ Should create presale
- ✓ Should create presale with native base token
- ✓ Should charge creation fee
- ✓ Should calculate hardcap correctly
- ✓ Should transfer sale token amount to presale
- ✓ Should fail with invalid fee profile
- ✓ Should fail with invalid sale token
- ✓ Should fail with invalid base token
- ✓ Should fail with invalid country code
- ✓ Should fail with invalid min-max buyer value
- ✓ Should fail with start time too low
- ✓ Should fail with invalid round1 start time
- ✓ Should fail with end time too low
- ✓ Should fail with invalid liquidity percent
- ✓ Should fail with locking period too low
- ✓ Should fail with invalid creation fee
- ✓ Should fail with invalid listing parameters
- ✓ Should fail without approving sale token amount
- ✓ Should fail with invalid softcap <> hardcap ratio
- ✓ Should fail with invalid amm parameters length

✓ Should fail with invalid amm index

✓ Should fail with dublicated amms on splitting

✓ Should fail with invalid splitting percentage

→ Discounted presale

✓ Should create presale

✓ Should charge lower creation fee

→ No hardcap presale

✓ Should create presale

✓ Should fail with invalid price increase

✓ Should fail with invalid creation fee

→ Custom presale

✓ Should create presale

✓ Should fail creation by non admin

✓ Should fail creation with invalid fee values

→ Admin functions

✓ Should set presale registry by admin

✓ Should set presale settings by admin

✓ Should set country list by admin

✓ Should fail setting presale registry by non-admin

✓ Should fail setting presale settings by non-admin

✓ Should fail setting country list by non-admin

→ PresaleGenerator

✓ Should initialize generator correctly

✓ Should set PresaleFactory by admin

✓ Should fail setting PresaleFactory by not-admin

✓ Should fail setting invalid PresaleFactory

✓ Should set diamond facets by admin

✓ Should fail setting diamond facets by non-admin

✓ Should disable setting diamond facets

✓ Should fail disabling setting diamond facets by not-admin

✓ Should fail creating presale by not-factory

→ PresaleLockForwarder

✓ Should create contract

✓ Should fail contract creation with invalid registry address

✓ Should initialize contract correctly

✓ Should fail locking liquidity with not registered presale

✓ Should set presale registry by admin

✓ Should fail setting presale registry by not-admin

✓ Should fail setting invalid presale registry

✓ Should state pool initialization on existing pool with funds

✓ Should not state pool initialization on existing pool without funds

→ Changing listed AMMs

✓ Should add amm by admin

✓ Should fail adding amm by not-admin

- ✓ Should fail adding amm with invalid locker contract
- ✓ Should fail adding amm dublicates
- ✓ Should activate/deactivate amm by admin
- ✓ Should fail activating/deactivating amm by not-admin
- ✓ Should fail toggle unlisted amm
- ✓ Should fail toggle with invalid status
- ✓ Should replace locker contract by admin
- ✓ Should fail replacing locker contract by not-admin
- ✓ Should fail replacing locker contract with invalid contract
- ✓ Should fail replacing locker contract with unlisted amm factory

→ Custom Presale

✓ Should initialize presale correctly

✓ Should charge custom fees on finalization

→ Discounted Presale

✓ Should create discounted presale

✓ Should initialize discounted presale correctly

✓ Should fail if custom whitelist is not set

✓ Should charge discounted sale token fee on finalization

✓ Should charge discounted base token fee on finalization

→ No Harcap Presale

✓ Should initialize presale correctly

→ Deposit

✓ Should not calculate sale token amount on deposit

✓ Should track deposited base token amount on deposit

✓ Should allow unlimited base token amount deposit

→ Finalization

✓ Should set correct token price on finalization

✓ Should init lp with correct amounts on finalization

✓ Should send correct base token amount to owner on finalization

✓ Should withdraw correct owed sale token amounts

→ Regular Presale

✓ Should initialize regular presale correctly

→ User deposit base token

✓ Should deposit in public round

✓ Should fail depositing while queued

✓ Should fail depositing while private round if not whitelisted

✓ Should deposit in private round if whitelisted

✓ Should fail depositing in public round with insufficient access tokens

✓ Should deposit in public round with sufficient access tokens

✓ Should deposit with native token

✓ Should not deposit more than max allowance

✓ Should fail depositing more than max allowance

✓ Should not deposit more than hardcap

✓ Should fail depositing if hardcap is already reached

✓ Should refund dust ether

✓ Should send custom base token to presale

✓ Should send native base token to presale

→ User withdraw base token

✓ Should withdraw total deposited base token on presale failure

✓ Should fail withdrawing before presale failed

✓ Should fail withdrawing without depositing

✓ Should fail withdrawing twice

✓ Should transfer base token back to user

✓ Should transfer native base token back to user

→ User emergency withdraw base token

✓ Should withdraw deposited base token when presale is active

✓ Should not charge fee on postponed presale

✓ Should not charge fee on presale fail

✓ Should fail emergency withdraw if presale ended successfully

→ Should charge fee on emergency withdraw when presale is not failed

✓ Should charge tier type NONE fee

✓ Should charge tier type ENTRY fee

✓ Should charge tier type MID fee

✓ Should charge tier type HIGH fee

→ User finalize presale with erc20 base

✓ Should finalize presale by owner in owner only time frame

✓ Should fail finalizing presale by anyone in owner only time frame

✓ Should fail finalizing failed presale

✓ Should finalize presale by anyone after owner only time frame

✓ Should set status to failed if pools have been initialized before

✓ Should charge base token fee on presale finalization

✓ Should charge sale token fee on presale finalization

✓ Should charge referrer fee amounts

✓ Should charge referral split fee amounts

✓ Should burn unsold sale tokens

✓ Should create new liquidity pool

✓ Should initialize liquidity pool with correct amounts

✓ Should send owner remaining base token amount on finalization

→ User finalize presale with native base

✓ Should finalize presale with native base tokens

✓ Should initialize pool with weth

✓ Should send owner remaining base token amount

→ User withdraw sale token

✓ Should emit withdraw event

✓ Should withdraw owed sale tokens

✓ Should fail withdrawing sale tokens if presale is not finalized

✓ Should fail withdrawing sale tokens if nothing deposited

→ Track presale status

- ✓ Should be in queued status after creation
- ✓ Should be in round 0 after start time if round 0 duration is set
- ✓ Should be in round 1 after start time if round 0 duration is not set
- ✓ Should be in round 1 after round 0
- ✓ Should be successful if hardcap is met
- ✓ Should be successful if softcap and end time is met
- ✓ Should be active if softcap is met but end time is not met
- ✓ Should be in failed state if softcap is not met but end time is met
- ✓ Should be in failed state if admin forced failure
- ✓ Should be finalized if presale was finalized (lp created)

→ PresaleRegistry

- ✓ Should initialize registry correctly
- ✓ Should set factory by owner
- ✓ Should fail setting factory by non-owner
- ✓ Should fail setting factory with invalid address
- ✓ Should fail registering presale by non factory
- ✓ Should emit register event
- ✓ Should register created presale correctly
- ✓ Should register correct presale type

→ PresaleSettings

- ✓ Should initialize settings correctly

✓ Should fail creating contract with invalid addresses

→ Set variables by owner

✓ Should set fee addresses

✓ Should fail setting invalid fee addresses

✓ Should set fees

✓ Should fail setting creation fee too low

✓ Should fail setting fees too high

✓ Should set emergency withdrawl fee for different tier levels

✓ Should fail setting emergency withdrawl fees too high

✓ Should fail setting invalid emergency withdrawl fees

✓ Should set staking contract

✓ Should fail setting staking contract with invalid address

✓ Should set default referrer

✓ Should fail setting zero address as default referrer

✓ Should set min locking liquidity percentage

✓ Should fail setting min locking liquidity percentage too high

✓ Should set min locking duration

✓ Should set owner finalize duration

✓ Should fail setting finalize owner duration too long

✓ Should set whitelist address

✓ Should fail setting invalid whitelist address

✓ Should set weth address

✓ Should fail setting invalid weth address

✓ Should set token vesting address

✓ Should fail setting invalid token vesting address

✓ Should set lock forwarder address

✓ Should fail setting invalid lock forwarder address

✓ Should add a custom fee profile

✓ Should fail adding invalid fee profile

✓ Should deactivate/activate existing fee profile

✓ Should fail deactivating invalid fee profile

→ Set variables by non-owner

✓ Should fail setting fee addresses

✓ Should fail setting fees

✓ Should fail setting default referrer

✓ Should fail setting min locking liquidity percentage

✓ Should fail setting min locking duration

✓ Should fail setting owner finalization duration

✓ Should fail setting whitelist address

✓ Should fail setting weth address

✓ Should fail setting token vesting address

✓ Should setting lock forwarder address

✓ Should fail adding a custom fee profile

✓ Should fail activating/deactivating a custom fee profile

→ Getter

✓ Should return default fee for no fee profile

✓ Should return relative fee for defined fee profiles

✓ Should fail getting fees for invalid fee profiles

→ Whitelist

✓ Should initialize whitelist correctly

→ Add single user

✓ Should add single user to whitelist by admin

✓ Should fail adding single user by not-admin

✓ Should fail adding zero address

✓ Should fail adding already whitelisted user again

→ Remove single user

✓ Should remove single user by admin

✓ Should fail removing single user by not-admin

✓ Should fail removing single user with zero address

✓ Should fail removing single not registered user

→ Add multiple users

✓ Should add multiple users by admin

✓ Should fail adding multiple users by not-admin

✓ Should fail adding zero address as user

✓ Should fail adding already registered user again

→ Remove multiple users

✓ Should remove multiple users by admin

✓ Should fail removing multiple users by not-admin

✓ Should fail removing multiple users with zero address

✓ Should fail removing multiple not registered users

301 passing (1m)

Coverage:

The code coverage results were obtained by running npx hardhat coverage in the ilov7-audit-main project. We found the following results :

- Statements Coverage : 98.69%
- Branches Coverage : 92.79%
- Functions Coverage : 100%
- Lines Coverage : 98.62%

6 Conclusion

In this audit, we examined the design and implementation of Unicrypt's ILO V7 and discovered several issues of varying severity. Unicrypt team addressed 0 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Unicrypt Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

7 Scope Files

7.1 Audit

Files	MD5 Hash
ilov7-audit-main/contracts/AdminRegistry.sol	ceab501034f449c0de8992c16f772b17
ilov7-audit-main/contracts/PresaleSettings.sol	6e887398776696aaf55936019b0e858c
ilov7-audit-main/contracts/PresaleRegistry.sol	0eb9653071ea5b8aee6e11f01d4a38bb
ilov7-audit-main/contracts/AdminRegistryImplementer.sol	a50c9ad8259f43e2f7ea838194c64ca3
ilov7-audit-main/contracts/PresaleFactory.sol	c371cc55fd004eecabd7d9ddf7edf7ec
ilov7-audit-main/contracts/PresaleGenerator.sol	c37d1ec8f9f432ce750322c48a1f1ff7
ilov7-audit-main/contracts/Whitelist.sol	4b450a5fe9fc11639f7d284bab8ac638
ilov7-audit-main/contracts/PresaleLockForwarder.sol	9e38cedd9a04884508ea5898dd010695
ilov7-audit-main/contracts/presale/PresaleDiamond.sol	9f5af6c47425a27d100fd7d34105ec36
ilov7-audit-main/contracts/presale/facets/DiamondLoupeFacet.sol	ec1e4a84b12a86faa7afa872689b80a6
ilov7-audit-main/contracts/presale/facets/PresaleLoupeFacet.sol	d3bb158338642ff43a537415744e8d2f
ilov7-audit-main/contracts/presale/facets/PresaleRestictedFacet.sol	cf64c09ad5537b43cdedd8409ed43f54
ilov7-audit-main/contracts/presale/facets/PresaleParticipantFacet.sol	11d1b037714a6184bb37825a32f00889

ilov7-audit-main/contracts/presale/facets/OwnershipFacet.sol	24a5d81d53605ddbe0d0fccbfc03b491
ilov7-audit-main/contracts/presale/interfaces/IPresaleDiamond.sol	2b407f0283a55ad734682affae76d220
ilov7-audit-main/contracts/presale/interfaces/IERC173.sol	01d6453755edd41e1ca03282a3fd9d6c
ilov7-audit-main/contracts/presale/interfaces/IDiamond.sol	8e17f7274793b9192b62de29951059ed
ilov7-audit-main/contracts/presale/interfaces/IPresaleLoupe.sol	12670d21091e79d8510c9820c8277573
ilov7-audit-main/contracts/presale/interfaces/IDiamondLoupe.sol	cb84fec62c9738d06639bcb5e9ee333a
ilov7-audit-main/contracts/presale/interfaces/IPresaleRestrictedFacet.sol	d929dbd182b55583031e9ff62ef5f410
ilov7-audit-main/contracts/presale/interfaces/IPresaleParticipantFacet.sol	fdb2904642cbad4ffb373164635ee77e
ilov7-audit-main/contracts/presale/libraries/LibPresaleStatus.sol	2b966b2eb3ae4a49414400644e9ff47a
ilov7-audit-main/contracts/presale/libraries/LibPresaleVesting.sol	e4350c74a28d07a6e82020c4481acabf
ilov7-audit-main/contracts/presale/libraries/LibPresaleBuyers.sol	b460852d4fcd90b70d119854d438cea6
ilov7-audit-main/contracts/presale/libraries/LibDiamond.sol	05a999f5fc844a7b47eb6f485d25d64f
ilov7-audit-main/contracts/presale/libraries/LibPresaleInfo.sol	0293c6d23b48b137c4b29b01430ba8cb
ilov7-audit-main/contracts/interfaces/IPresaleGenerator.sol	226c5618ddbdbb3a579afcddaa80e77d

ilov7-audit-main/contracts/interfaces/IPresaleLockForwarder.sol	a31b35e4f1695021181755c715795c29
ilov7-audit-main/contracts/interfaces/IWhitelist.sol	11d4de500695f5a86eb6c4ce249b00e0
ilov7-audit-main/contracts/interfaces/IAdminRegistry.sol	2c24161ef45eeffcb73b9ed3141caff4
ilov7-audit-main/contracts/interfaces/IPresaleFactory.sol	ee42130062108a0de2444e89880f56e1
ilov7-audit-main/contracts/interfaces/IPresaleSettings.sol	80aacae1f9a2e614d99685bf09c1841e
ilov7-audit-main/contracts/interfaces/IWETH.sol	4c49fc788e6e2fbd8b25c1668573d1c1
ilov7-audit-main/contracts/interfaces/ITokenVesting.sol	e208e353fc117f5c77a156a7250ce2ef
ilov7-audit-main/contracts/interfaces/IPresaleRegistry.sol	fd1fe556480ab0288731a764e4bcf497
ilov7-audit-main/contracts/interfaces/ICountryList.sol	d3a16b2224304344a88eec2913247df3
ilov7-audit-main/contracts/interfaces/AMM/ILPLocker.sol	45eefdf29b33d90bdfc9692db6b20e3b
ilov7-audit-main/contracts/interfaces/AMM/IAMMFactory.sol	f0bf0dbef43963127e2418e49b9bb41e
ilov7-audit-main/contracts/interfaces/AMM/ILiquidityPool.sol	17b7f21f87dccc460719df89b37b3b03
ilov7-audit-main/contracts/libraries/FullMath.sol	20819ad28cfd4d57bf2bd2e13985b85f
ilov7-audit-main/contracts/libraries/TransferHelper.sol	98837f658a121cc8196056425b5b2483

7.2 Re-Audit

Files	MD5 Hash
ilo7-audit/contracts/Whitelist.sol	4b450a5fe9fc11639f7d284bab8ac638
ilo7-audit/contracts/PresaleFactory.sol	b5ae673ffba1f2f491af767c3e6d10d0
ilo7-audit/contracts/AdminRegistryImplementer.sol	a50c9ad8259f43e2f7ea838194c64ca3
ilo7-audit/contracts/PresaleGenerator.sol	c37d1ec8f9f432ce750322c48a1f1ff7
ilo7-audit/contracts/PresaleRegistry.sol	0eb9653071ea5b8aee6e11f01d4a38bb
ilo7-audit/contracts/AdminRegistry.sol	ceab501034f449c0de8992c16f772b17
ilo7-audit/contracts/PresaleLockForwarder.sol	9e38cedd9a04884508ea5898dd010695
ilo7-audit/contracts/PresaleSettings.sol	b62681ab3201a7530f465875dcc8882e
ilo7-audit/contracts/interfaces/IPresaleFactory.sol	ee42130062108a0de2444e89880f56e1
ilo7-audit/contracts/interfaces/IWETH.sol	4c49fc788e6e2fbd8b25c1668573d1c1
ilo7-audit/contracts/interfaces/IWhitelist.sol	11d4de500695f5a86eb6c4ce249b00e0
ilo7-audit/contracts/interfaces/IPresaleRegistry.sol	fd1fe556480ab0288731a764e4bcf497
ilo7-audit/contracts/interfaces/ICountryList.sol	d3a16b2224304344a88eec2913247df3
ilo7-audit/contracts/interfaces/IPresaleSettings.sol	5fa870f0f2dd57caf688aaaf7ffefb16
ilo7-audit/contracts/interfaces/IAdminRegistry.sol	2c24161ef45eeffcb73b9ed3141caff4
ilo7-audit/contracts/interfaces/IPresaleLockForwarder.sol	a31b35e4f1695021181755c715795c29

ilo7-audit/contracts/interfaces/ITokenVesting.sol	e208e353fc117f5c77a156a7250ce2ef
ilo7-audit/contracts/interfaces/IPresaleGenerator.sol	226c5618ddbdbb3a579afcddaa80e77d
ilo7-audit/contracts/interfaces/AMM/IAMMFactory.sol	f0bf0dbef43963127e2418e49b9bb41e
ilo7-audit/contracts/interfaces/AMM/ILiquidityPool.sol	17b7f21f87dccc460719df89b37b3b03
ilo7-audit/contracts/interfaces/AMM/ILPLocker.sol	45eefdf29b33d90bdfc9692db6b20e3b
ilo7-audit/contracts/libraries/TransferHelper.sol	98837f658a121cc8196056425b5b2483
ilo7-audit/contracts/libraries/FullMath.sol	20819ad28cfd4d57bf2bd2e13985b85f
ilo7-audit/contracts/presale/PresaleDiamond.sol	9f5af6c47425a27d100fd7d34105ec36
ilo7-audit/contracts/presale/interfaces/IDiamondLoupe.sol	cb84fec62c9738d06639bcb5e9ee333a
ilo7-audit/contracts/presale/interfaces/IPresaleLoupe.sol	12670d21091e79d8510c9820c8277573
ilo7-audit/contracts/presale/interfaces/IERC173.sol	01d6453755edd41e1ca03282a3fd9d6c
ilo7-audit/contracts/presale/interfaces/IPresaleDiamond.sol	2b407f0283a55ad734682affae76d220
ilo7-audit/contracts/presale/interfaces/IPresaleParticipantFacet.sol	fdb2904642cbad4ffb373164635ee77e
ilo7-audit/contracts/presale/interfaces/IPresaleRestrictedFacet.sol	d929dbd182b55583031e9ff62ef5f410

ilo7-audit/contracts/presale/interfaces/IDiamond.sol	8e17f7274793b9192b62de29951059ed
ilo7-audit/contracts/presale/libraries/LibPresaleVesting.sol	e4350c74a28d07a6e82020c4481acabf
ilo7-audit/contracts/presale/libraries/LibPresaleBuyers.sol	b460852d4fcd90b70d119854d438cea6
ilo7-audit/contracts/presale/libraries/LibPresaleStatus.sol	2b966b2eb3ae4a49414400644e9ff47a
ilo7-audit/contracts/presale/libraries/LibPresaleInfo.sol	0293c6d23b48b137c4b29b01430ba8cb
ilo7-audit/contracts/presale/libraries/LibDiamond.sol	05a999f5fc844a7b47eb6f485d25d64f
ilo7-audit/contracts/presale/facets/OwnershipFacet.sol	24a5d81d53605ddbe0d0fccbfc03b491
ilo7-audit/contracts/presale/facets/PresaleRestictedFacet.sol	ee46847f2c6c6cae81bc6ae5c4111c28
ilo7-audit/contracts/presale/facets/PresaleParticipantFacet.sol	11d1b037714a6184bb37825a32f00889
ilo7-audit/contracts/presale/facets/DiamondLoupeFacet.sol	ec1e4a84b12a86faa7afa872689b80a6
ilo7-audit/contracts/presale/facets/PresaleLoupeFacet.sol	d3bb158338642ff43a537415744e8d2f

8 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com