// HALBORN

LID CONTRACTS

Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: **09.22-30.2020**
Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	9/30/2020	Nishit Majitia

CONTACTS

CONTACT	COMPANY	EMAIL
STEVEN WALBROEHL	Halborn	Steven.Walbroehl@halborn.com
ROB BEHNKE	Halborn	Rob.Behnke@halborn.com
Nishit Majithia	Halborn	HalbornNishit.Majithia@halborn.com

1.1 INTRODUCTION

LID engaged Halborn to conduct a security assessment on their all smart contract beginning on September 22th, 2020 and ending September 30th 2020. The security assessment was scoped to the contract LidCertifiedPresale, LidCertifiedPresaleTimer, LidDaoLock, LidPromoFund, LidStaking, LidStakingV2, LidStakingFund, LidTeamLock, LidToken, LidVotingRights and Migrations and an audit of the security risk and implications regarding the changes introduced by the development team at LID prior to its production release shortly following the assessments deadline.

The contract scoped in this assessment is focused to the contracts LidStakingV2 and LidDaoFund. LID is a DAO token which governs the treasury that covers traders in the unlikely event of a black swan, where margin collateral is not sufficient to cover open margin positions.

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices for secure smart contract development. Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

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

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

1.3 SCOPE

IN-SCOPE:
LID contracts (commit - 1a48a589588dedbfc3a19e7ef82a199214dbd0ff).

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository or imported by LID contracts, economic attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	3	1

SECURITY ANALYSIS	RISK LEVEL
DoS WITH BLOCK GAS LIMIT	Medium
UNINITIALIZED STATE VARIABLE	Medium
STATIC ANALYSIS REPORT	Medium
STATE VARIABLE VISIBILITY NOT SET	Low
NO RETURN VALUE	Informational
SECURITY FUZZING RESULTS	Informational

FINDINGS & TECH DETAILS

3.1 DoS WITH BLOCK GAS LIMIT - MEDIUM

Description

When smart contracts are deployed or functions inside them are called, the execution of these actions always requires a certain amount of gas, based on how much computation is needed to complete them. The Ethereum network specifies a block gas limit and the sum of all transactions included in a block cannot exceed the threshold.

Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition.

A situation in which the block gas limit can be an issue is in sending funds to an array of addresses. Even without any malicious intent, this can easily go wrong. Just by having too large an array of users to pay can max out the gas limit and prevent the transaction from ever succeeding.

Code Location

LidCertifiedPresale.sol: Line #126, Line #130, Line #146, Line #150, Line #184, Line #196

LidStaking.sol: Line #121, Line #218 LidStakingV2.sol: Line #99

```
125         uniswapEthBP = 7500; //75%
126         for (uint i = 0; i < _etherPools.length; ++i) {
127             etherPools.push(_etherPools[i]);
128         }
129         uint totalEtherPoolsBP = uniswapEthBP;
```

```
129         uint totalEtherPoolsBP = uniswapEthBP;
130         for (uint i = 0; i < _etherPoolBPs.length; ++i) {
131             etherPoolBPs.push(_etherPoolBPs[i]);
132             totalEtherPoolsBP = totalEtherPoolsBP.add(_etherPoolBPs[i]);
133         }
```

```
145         presaleTokenBP = 4000;
146         for (uint i = 0; i < _tokenPools.length; ++i) {
147             tokenPools.push(_tokenPools[i]);
148         }
149         uint totalTokenPoolBPs = uniswapTokenBP.add(presaleTokenBP);
150         for (uint i = 0; i < _tokenPoolBPs.length; ++i) {
151             tokenPoolBPs.push(_tokenPoolBPs[i]);
152             totalTokenPoolBPs = totalTokenPoolBPs.add(_tokenPoolBPs[i]);
153         }
```

```
195         hasSentEther = true;
196         for (uint i = 0; i < etherPools.length; ++i) {
197             etherPools[i].transfer(
198                 totalEth.mulBP(etherPoolBPs[i])
199             );
200         }
```


```
195         hasSentEther = true;
196         for (uint i = 0; i < etherPools.length; ++i) {
197             etherPools[i].transfer(
198                 totalEth.mulBP(etherPoolBPs[i])
199             );
200         }
```


Dynamic arrays (_etherPools, _etherPoolBPs, _tokenPools, _tokenPoolBPs) are in control of the caller. Also, in the code I can see totalEtherPoolsBP == 10000 and uniswapEthBP = 7500 for setEtherPools() so only 2500 possible entries can come if _etherPoolBPs value is >0 but since _etherPoolBPs are uint[] it can have value 0 for any _etherPools address. So, this array is not restricted to maximum 2500 in size.

Same case with setTokenPools() function. It also has condition like totalEtherPoolsBP == 10000 and uniswapTokenBP = 1600 plus presaleTokenBP = 4000. So in this case also, dynamic array _tokenPools can go maximum up to 4400 in length if _tokenPoolBPs has value >0. But since _tokenPoolBPs is uint[], it can have value 0 for any _tokenPools address. So, this array is also not restricted to maximum 4400 in size.

In case of LidStaking contract, dynamic array stakeHandlers value can be increase when someone register as stakeHandler. So, this array can also go up to very large uint value.

In case of LidTeamLock contract, while initialization and resetting the team _teamMemberAddresses and _teamMemberBPs which are dynamic arrays are in control of caller. Though there is condition totalTeamBP == 10000 in initialize() which can limit the size of _teamMemberAddresses and _teamMemberBPs to 10000 if each value of _teamMemberBPs array is >0, but since it is uint[], the size of both the dynamic array is not restricted to 10000. This condition to restricting totalTeamBP to 10000 is missing from resetTeam() function. So, size of data structure that may grow unboundedly.

Recommendation:
Actions that require looping across the entire data structure should be avoided. If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions.
In this case, if you want _etherPooBPs and _tokenPoolBPs ‘s values should be >0 then the size of _etherPools and _tokenPools will be restricted.

3.2 UNINITIALIZED STATE VARIABLE - MEDIUM

Description:
State variable is being use without getting initialize in the contract.

Code Location:
LidToken.sol: Line #125, Line #137

be used directly without being initialized inside transfer() and transferFrom() functions.

Recommendation:
In LidToken.sol contract, transfer() and transferFrom() functions

are public functions and can be callable by anyone. It will fail the execution of both of these methods if map `toOnlyTaxExempt` is not set manually by the contract owner. Remediation of this issue should be adding initialization to map `toOnlyTaxExempt` in `initialize()` call while deploying the contract or initialize it while declaring it.

3.3 STATIC ANALYSIS REPORT - MEDIUM

Description:

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

Results

```
INFO:Detectors:
UniswapV2Migrator._safeTransferETH(address,uint256) (uniswapV2Periphery/UniswapV2Migrator.sol#27-30) sends eth to arbitrary user
        Dangerous calls:
        - (success) = to.call.value(value)(new bytes(0)) (uniswapV2Periphery/UniswapV2Migrator.sol#28)
UniswapV2Router01.swapExactETHForTokens(uint256,address[],address,uint256) (uniswapV2Periphery/UniswapV2Router01.sol#215-227) sends eth to arbitrary user
        Dangerous calls:
        - WETH.deposit.value(amounts[0])() (uniswapV2Periphery/UniswapV2Router01.sol#224)
LidCertifiedPresale.emergencyEthWithdrawl() (LidCertifiedPresale.sol#209-212) sends eth to arbitrary user
        Dangerous calls:
        - msg.sender.transfer(address(this).balance) (LidCertifiedPresale.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
```

```
INFO:Detectors:
ERC20Pausable._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Pausable.sol#40) shadows:
        - Pausable._____gap (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#79)
        - PauserRole._____gap (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#49)
        - ERC20._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#233)
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
ERC20Burnable._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol#30) shadows:
        - ERC20._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#233)
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
LidStakingV2.lidToken (LidStakingV2.sol#27) shadows:
        - LidStaking.lidToken (LidStaking.sol#19)
PauserRole._____gap (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#49) shadows:
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
ERC20Mintable._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Mintable.sol#30) shadows:
        - MinterRole._____gap (@openzeppelin/contracts-ethereum-package/contracts/access/roles/MinterRole.sol#49)
        - ERC20._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#233)
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
ReentrancyGuard._____gap (@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol#41) shadows:
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
Pausable._____gap (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#79) shadows:
        - PauserRole._____gap (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#49)
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
MinterRole._____gap (@openzeppelin/contracts-ethereum-package/contracts/access/roles/MinterRole.sol#49) shadows:
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
LidToken._name (LidToken.sol#41) shadows:
        - ERC20Detailed._name (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#10)
ERC20Detailed._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#56) shadows:
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
Ownable._____gap (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#80) shadows:
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
ERC20._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#233) shadows:
        - Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
```

```
INFO:Detectors:
LidToken.toOnlyTaxExempt (LidToken.sol#39) is never initialized. It is used in:
        - LidToken.transfer(address,uint256) (LidToken.sol#120-130)
        - LidToken.transferFrom(address,address,uint256) (LidToken.sol#132-151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
```

```
INFO:Detectors:
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) performs a multiplication on the result of a division:
        -multiplier = bal.div(1000000000000000000) (LidCertifiedPresaleTimer.sol#41)
        -startTime.add(baseTimer).add(deltaTimer.mul(multiplier)) (LidCertifiedPresaleTimer.sol#53-57)
LidCertifiedPresale.calculateRatePerEth() (LidCertifiedPresale.sol#287-289) performs a multiplication on the result of a division:
        -totalTokens.div(10 ** 18).mul(multiplierPrice).add(startingPrice) (LidCertifiedPresale.sol#288)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
UniswapV2Migrator._safeApprove(address,address,uint256) (uniswapV2Periphery/UniswapV2Migrator.sol#17-20) uses a dangerous strict equality:
        - require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))),APPROVE_FAILED) (uniswapV2Periphery/UniswapV2Migrator.sol#19)
UniswapV2Migrator._safeTransfer(address,address,uint256) (uniswapV2Periphery/UniswapV2Migrator.sol#22-25) uses a dangerous strict equality:
        - require(bool,string)(success && (data.length == 0 || abi.decode(data,(bool))),TRANSFER_FAILED) (uniswapV2Periphery/UniswapV2Migrator.sol#24)
LidTeamLock.startRelease() (LidTeamLock.sol#82-89) uses a dangerous strict equality:
        - require(bool,string)(releaseStart == 0,Has already started.) (LidTeamLock.sol#83)
LidCertifiedPresale.calculateReedemable(address) (LidCertifiedPresale.sol#272-285) uses a dangerous strict equality:
        - finalEndTime == 0 (LidCertifiedPresale.sol#273)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```

```
INFO:Detectors:
Contract locking ether found in :
        Contract LidStakingFund (LidStakingFund.sol#8-41) has payable functions:
         - LidStakingFund.fallback() (LidStakingFund.sol#28)
        But does not have a function to withdraw the ether
Contract locking ether found in :
        Contract LidDaoLock (LidDaoLock.sol#10-71) has payable functions:
         - LidDaoLock.fallback() (LidDaoLock.sol#46)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

```
INFO:Detectors:
Contract locking ether found in :
        Contract LidStakingFund (LidStakingFund.sol#8-41) has payable functions:
         - LidStakingFund.fallback() (LidStakingFund.sol#28)
        But does not have a function to withdraw the ether
Contract locking ether found in :
        Contract LidDaoLock (LidDaoLock.sol#10-71) has payable functions:
         - LidDaoLock.fallback() (LidDaoLock.sol#46)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

```
INFO:Detectors:
Contract locking ether found in :
        Contract LidStakingFund (LidStakingFund.sol#8-41) has payable functions:
         - LidStakingFund.fallback() (LidStakingFund.sol#28)
        But does not have a function to withdraw the ether
Contract locking ether found in :
        Contract LidDaoLock (LidDaoLock.sol#10-71) has payable functions:
         - LidDaoLock.fallback() (LidDaoLock.sol#46)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

```
INFO:Detectors:
LidStakingV2.stakeValueAt(address,uint256)._owner (LidStakingV2.sol#49) shadows:
        - Ownable._owner (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#17) (state variable)
LidCertifiedPresaleTimer.initialize(uint256,uint256,uint256,address).owner (LidCertifiedPresaleTimer.sol#20) shadows:
        - Ownable.owner() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#32-34) (function)
LidCertifiedPresale.initialize(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,address,LidCertifiedPresaleTimer,ILidCertifiableToken).owner (LidCer
tifiedPresale.sol#85) shadows:
        - Ownable.owner() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#32-34) (function)
LidToken.initialize(string,string,uint8,address,uint256,uint256,address,LidStaking,LidCertifiedPresale).name (LidToken.sol#49) shadows:
        - LidToken.name() (LidToken.sol#116-118) (function)
        - ERC20Detailed.name() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#28-30) (function)
LidToken.initialize(string,string,uint8,address,uint256,uint256,address,LidStaking,LidCertifiedPresale).symbol (LidToken.sol#49) shadows:
        - ERC20Detailed.symbol() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#36-38) (function)
LidToken.initialize(string,string,uint8,address,uint256,uint256,address,LidStaking,LidCertifiedPresale).decimals (LidToken.sol#49) shadows:
        - ERC20Detailed.decimals() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#52-54) (function)
LidToken.initialize(string,string,uint8,address,uint256,uint256,address,LidStaking,LidCertifiedPresale).owner (LidToken.sol#50) shadows:
        - Ownable.owner() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#32-34) (function)
LidStaking.initialize(uint256,uint256,uint256,uint256,address,ILidCertifiableToken).owner (LidStaking.sol#60) shadows:
        - Ownable.owner() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#32-34) (function)
ERC20Detailed.initialize(string,string,uint8).name (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#19) shadows:
        - ERC20Detailed.name() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#28-30) (function)
ERC20Detailed.initialize(string,string,uint8).symbol (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#19) shadows:
        - ERC20Detailed.symbol() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#36-38) (function)
ERC20Detailed.initialize(string,string,uint8).decimals (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#19) shadows:
        - ERC20Detailed.decimals() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#52-54) (function)
LidDaoLock.initialize(uint256,uint256,address,ILidCertifiableToken).owner (LidDaoLock.sol#33) shadows:
        - Ownable.owner() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#32-34) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

```
INFO:Detectors:
LidStakingV2._superRemoveStake(uint256) (LidStakingV2.sol#91-105) has external calls inside a loop: stakeHandlers[i].handleUnstake(msg.sender,_amount,stakeValue[ms
g.sender]) (LidStakingV2.sol#100)
UniswapV2Router01._swap(uint256[],address[],address) (uniswapV2Periphery/UniswapV2Router01.sol#181-190) has external calls inside a loop: IUniswapV2Pair(pairFor(in
put,output)).swap(amount0Out,amount1Out,to,new bytes(0)) (uniswapV2Periphery/UniswapV2Router01.sol#188)
LidCertifiedPresale.issueTokens() (LidCertifiedPresale.sol#180-190) has external calls inside a loop: token.mint(tokenPools[i],totalTokens.mulBP(tokenPoolBPs[i]))
(LidCertifiedPresale.sol#185-188)
LidCertifiedPresale.sendEther() (LidCertifiedPresale.sol#192-207) has external calls inside a loop: etherPools[i].transfer(totalEth.mulBP(etherPoolBPs[i])) (LidCer
tifiedPresale.sol#197-199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in LidStaking.registerAndStake(uint256,address) (LidStaking.sol#77-95):
        External calls:
        - require(bool,string)(lidToken.transferFrom(msg.sender,referrer,registrationFeeWithReferrer),Stake failed due to failed referral transfer.) (LidStaking.so
l#89)
        State variables written after the call(s):
        - accountReferrals[referrer] = accountReferrals[referrer].add(1) (LidStaking.sol#90)
Reentrancy in LidToken.transferFrom(address,address,uint256) (LidToken.sol#132-151):
        External calls:
        - _transferWithTax(sender,recipient,amount) (LidToken.sol#134-140)
                - lidStaking.handleTaxDistribution(tax) (LidToken.sol#176)
        State variables written after the call(s):
        - approve(msg.sender,allowance(sender,msg.sender).sub(amount,Transfer amount exceeds allowance)) (LidToken.sol#142-149)
                - _allowances[owner][spender] = amount (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#218)
Reentrancy in LidStaking.unstake(uint256) (LidStaking.sol#107-127):
        External calls:
        - withdraw(dividendsOf(msg.sender)) (LidStaking.sol#111)
                - lidToken.transfer(msg.sender,amount) (LidStaking.sol#132)
        State variables written after the call(s):
        - _increaseProfitPerShare(tax) (LidStaking.sol#118)
                - emptyStakeTokens = 0 (LidStaking.sol#230)
                - emptyStakeTokens = emptyStakeTokens.add(amount) (LidStaking.sol#234)
        - totalStaked = totalStaked.sub(amount) (LidStaking.sol#113)
        - totalStakers = totalStakers.sub(1) (LidStaking.sol#112)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Reentrancy in LidStakingV2._superRemoveStake(uint256) (LidStakingV2.sol#91-105):
        External calls:
        - require(bool,string)(lidToken.transferFrom(address(this),msg.sender,earnings),Unstake failed due to failed transfer.) (LidStakingV2.sol#103)
        Event emitted after the call(s):
        - OnUnstake(msg.sender,_amount,tax) (LidStakingV2.sol#104)
Reentrancy in LidStaking.distribute(uint256) (LidStaking.sol#144-153):
        External calls:
        - require(bool,string)(lidToken.transferFrom(msg.sender,address(this),amount),Distribution failed due to failed transfer.) (LidStaking.sol#148-151)
        Event emitted after the call(s):
        - OnDistribute(msg.sender,amount) (LidStaking.sol#152)
Reentrancy in LidStaking.registerAndStake(uint256,address) (LidStaking.sol#77-95):
        External calls:
        - distribute(registrationFeeWithoutReferrer) (LidStaking.sol#84)
                - require(bool,string)(lidToken.transferFrom(msg.sender,address(this),amount),Distribution failed due to failed transfer.) (LidStaking.sol#148-151)
        - require(bool,string)(lidToken.transferFrom(msg.sender,referrer,registrationFeeWithReferrer),Stake failed due to failed referral transfer.) (LidStaking.so
l#89)
        - stake(finalAmount) (LidStaking.sol#94)
                - require(bool,string)(lidToken.transferFrom(msg.sender,address(this),amount),Stake failed due to failed transfer.) (LidStaking.sol#103)
                - stakeHandlers[i].handleStake(msg.sender,stakeAmount,stakeValue[msg.sender]) (LidStaking.sol#219)
        Event emitted after the call(s):
        - OnStake(msg.sender,amount,tax) (LidStaking.sol#104)
                - stake(finalAmount) (LidStaking.sol#94)
```

```
Reentrancy in LidStaking.reinvest(uint256) (LidStaking.sol#136-142):
        External calls:
        - tax = _addStake(amount) (LidStaking.sol#140)
                - stakeHandlers[i].handleStake(msg.sender,stakeAmount,stakeValue[msg.sender]) (LidStaking.sol#219)
        Event emitted after the call(s):
        - OnReinvest(msg.sender,amount,tax) (LidStaking.sol#141)
Reentrancy in LidStaking.stake(uint256) (LidStaking.sol#97-105):
        External calls:
        - tax = _addStake(amount) (LidStaking.sol#102)
                - stakeHandlers[i].handleStake(msg.sender,stakeAmount,stakeValue[msg.sender]) (LidStaking.sol#219)
        - require(bool,string)(lidToken.transferFrom(msg.sender,address(this),amount),Stake failed due to failed transfer.) (LidStaking.sol#103)
        Event emitted after the call(s):
        - OnStake(msg.sender,amount,tax) (LidStaking.sol#104)
Reentrancy in LidToken.transferFrom(address,address,uint256) (LidToken.sol#132-151):
        External calls:
        - _transferWithTax(sender,recipient,amount) (LidToken.sol#134-140)
                - lidStaking.handleTaxDistribution(tax) (LidToken.sol#176)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#219)
                - approve(msg.sender,allowance(sender,msg.sender).sub(amount,Transfer amount exceeds allowance)) (LidToken.sol#142-149)
Reentrancy in LidStaking.unstake(uint256) (LidStaking.sol#107-127):
        External calls:
        - withdraw(dividendsOf(msg.sender)) (LidStaking.sol#111)
                - lidToken.transfer(msg.sender,amount) (LidStaking.sol#132)
        - require(bool,string)(lidToken.transferFrom(address(this),msg.sender,earnings),Unstake failed due to failed transfer.) (LidStaking.sol#125)
        Event emitted after the call(s):
        - OnUnstake(msg.sender,amount,tax) (LidStaking.sol#126)
Reentrancy in LidStaking.withdraw(uint256) (LidStaking.sol#129-134):
        External calls:
        - lidToken.transfer(msg.sender,amount) (LidStaking.sol#132)
        Event emitted after the call(s):
        - OnWithdraw(msg.sender,amount) (LidStaking.sol#133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
INFO:Detectors:
Initializable.isConstructor() (@openzeppelin/upgrades/contracts/Initializable.sol#48-58) uses assembly
        - INLINE ASM (@openzeppelin/upgrades/contracts/Initializable.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
LidStaking.stake(uint256) (LidStaking.sol#97-105) compares to a boolean constant:
        -require(bool,string)(stakerIsRegistered[msg.sender] == true,Must be registered to stake.) (LidStaking.sol#98)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
INFO:Detectors:
Low level call in UniswapV2Migrator._safeApprove(address,address,uint256) (uniswapV2Periphery/UniswapV2Migrator.sol#17-20):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR_APPROVE,to,value)) (uniswapV2Periphery/UniswapV2Migrator.sol#18)
Low level call in UniswapV2Migrator._safeTransfer(address,address,uint256) (uniswapV2Periphery/UniswapV2Migrator.sol#22-25):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR_TRANSFER,to,value)) (uniswapV2Periphery/UniswapV2Migrator.sol#23)
Low level call in UniswapV2Migrator._safeTransferETH(address,uint256) (uniswapV2Periphery/UniswapV2Migrator.sol#27-30):
        - (success) = to.call.value(value)(new bytes(0)) (uniswapV2Periphery/UniswapV2Migrator.sol#28)
Low level call in UniswapV2Router01._safeTransfer(address,address,uint256) (uniswapV2Periphery/UniswapV2Router01.sol#15-18):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR_TRANSFER,to,value)) (uniswapV2Periphery/UniswapV2Router01.sol#16)
Low level call in UniswapV2Router01._safeTransferFrom(address,address,address,uint256) (uniswapV2Periphery/UniswapV2Router01.sol#19-22):
        - (success,data) = token.call(abi.encodeWithSelector(SELECTOR_TRANSFER_FROM,from,to,value)) (uniswapV2Periphery/UniswapV2Router01.sol#20)
Low level call in UniswapV2Router01._safeTransferETH(address,uint256) (uniswapV2Periphery/UniswapV2Router01.sol#23-26):
        - (success) = to.call.value(value)(new bytes(0)) (uniswapV2Periphery/UniswapV2Router01.sol#24)
Low level call in ExampleFlashSwap.uniswapV2Call(address,uint256,uint256,bytes) (uniswapV2Periphery/ExampleFlashSwap.sol#27-65):
        - (success) = sender.call.value(amountReceived - amountRequired)(new bytes(0)) (uniswapV2Periphery/ExampleFlashSwap.sol#54)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
LidTeamLock.claimLid(uint256) (LidTeamLock.sol#59-69) uses timestamp for comparisons
        Dangerous comparisons:
        - lidToken.balanceOf(address(this)) < toClaim (LidTeamLock.sol#66)
LidTeamLock.startRelease() (LidTeamLock.sol#82-89) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(releaseStart == 0,Has already started.) (LidTeamLock.sol#83)
LidTeamLock.getCurrentCycleCount() (LidTeamLock.sol#109-112) uses timestamp for comparisons
        Dangerous comparisons:
        - now <= releaseStart (LidTeamLock.sol#110)
LidCertifiedPresaleTimer.isStarted() (LidCertifiedPresaleTimer.sol#34-36) uses timestamp for comparisons
        Dangerous comparisons:
        - (startTime != 0 && now > startTime) (LidCertifiedPresaleTimer.sol#35)
ExampleOracleSimple.constructor(address,address) (uniswapV2Periphery/ExampleOracleSimple.sol#20-30) uses timestamp for comparisons
        Dangerous comparisons:
        - assert(bool)(blockTimestampLast != 0) (uniswapV2Periphery/ExampleOracleSimple.sol#29)
ExampleOracleSimple.update() (uniswapV2Periphery/ExampleOracleSimple.sol#32-55) uses timestamp for comparisons
        Dangerous comparisons:
        - assert(bool)(timeElapsed >= PERIOD) (uniswapV2Periphery/ExampleOracleSimple.sol#35)
        - blockTimestampLastFromPair != blockTimestamp (uniswapV2Periphery/ExampleOracleSimple.sol#43)
LidCertifiedPresale.calculateReedemable(address) (LidCertifiedPresale.sol#272-285) uses timestamp for comparisons
        Dangerous comparisons:
        - finalEndTime == 0 (LidCertifiedPresale.sol#273)
        - totalRedeemable >= earnedLid (LidCertifiedPresale.sol#279)
LidCertifiedPresale._isPresaleEnded() (LidCertifiedPresale.sol#295-299) uses timestamp for comparisons
        Dangerous comparisons:
        - ((timer.isStarted() && (now > timer.getEndTime(address(this).balance)))) (LidCertifiedPresale.sol#296-298)
LidDaoLock.claimLid() (LidDaoLock.sol#48-56) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(releaseStart != 0,Has not yet started.) (LidDaoLock.sol#49)
        - lidToken.balanceOf(address(this)) < toClaim (LidDaoLock.sol#53)
LidDaoLock.startRelease(address) (LidDaoLock.sol#58-64) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(releaseStart == 0,Has already started.) (LidDaoLock.sol#59)
LidDaoLock.getCurrentCycleCount() (LidDaoLock.sol#66-69) uses timestamp for comparisons
        Dangerous comparisons:
        - now <= releaseStart (LidDaoLock.sol#67)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Pragma version>=0.5.0 (@uniswap/v2-core/contracts/interfaces/IUniswapV2Callee.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/access/Roles.sol#1) allows old versions
Pragma version>=0.5.0 (@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Pausable.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol#1) allows old versions
Pragma version>=0.4.24<0.7.0 (@openzeppelin/upgrades/contracts/Initializable.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Mintable.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/GSN/Context.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/access/roles/MinterRole.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#1) allows old versions
Pragma version>=0.4.21<0.7.0 (Migrations.sol#2) allows old versions
Pragma version>=0.5.0 (@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#1) allows old versions
Pragma version^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol#1) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

FINDINGS & TECH DETAILS

```
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.5.16', '=0.5.16', '>=0.4.21<0.7.0', '>=0.4.24<0.7.0', '>=0.5.0', '^0.5.0']
        - 0.5.16 (LidVotingRights.sol#1)
        - >=0.5.0 (@uniswap/v2-core/contracts/interfaces/IUniswapV2Callee.sol#1)
        - =0.5.16 (uniswapV2Periphery/UniswapV2Migrator.sol#1)
        - 0.5.16 (interfaces/IStakeHandler.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/access/Roles.sol#1)
        - >=0.5.0 (@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol#1)
        - =0.5.16 (uniswapV2Periphery/interfaces/IUniswapV2Library.sol#1)
        - =0.5.16 (uniswapV2Periphery/libraries/UQ112x112.sol#1)
        - 0.5.16 (LidTeamLock.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Pausable.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol#1)
        - 0.5.16 (LidStakingV2.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol#1)
        - >=0.4.24<0.7.0 (@openzeppelin/upgrades/contracts/Initializable.sol#1)
        - 0.5.16 (LidPromoFund.sol#1)
        - 0.5.16 (LidCertifiedPresaleTimer.sol#1)
        - =0.5.16 (uniswapV2Periphery/interfaces/V1/IUniswapV1Factory.sol#1)
        - =0.5.16 (uniswapV2Periphery/UniswapV2Router01.sol#1)
        - 0.5.16 (library/BasisPoints.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#1)
        - 0.5.16 (LidStakingFund.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Mintable.sol#1)
        - 0.5.16 (interfaces/ILidCertifiableToken.sol#1)
        - =0.5.16 (uniswapV2Periphery/interfaces/V1/IUniswapV1Exchange.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol#1)
        - =0.5.16 (uniswapV2Periphery/ExampleFlashSwap.sol#1)
        - =0.5.16 (uniswapV2Periphery/UniswapV2Library.sol#1)
        - =0.5.16 (uniswapV2Periphery/ExampleOracleSimple.sol#1)
        - 0.5.16 (LidCertifiedPresale.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/GSN/Context.sol#1)
        - =0.5.16 (uniswapV2Periphery/interfaces/IUniswapV2Router01.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/access/roles/MinterRole.sol#1)
        - 0.5.16 (LidToken.sol#1)
        - 0.5.16 (LidStaking.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#1)
        - >=0.4.21<0.7.0 (Migrations.sol#2)
        - =0.5.16 (uniswapV2Periphery/interfaces/IUniswapV2Migrator.sol#1)
        - =0.5.16 (uniswapV2Periphery/interfaces/IWETH.sol#1)
        - >=0.5.0 (@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#1)
        - ^0.5.0 (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol#1)
        - 0.5.16 (LidDaoLock.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
INFO:Detectors:
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - bal <= 10000000000000000000000 (LidCertifiedPresaleTimer.sol#44)
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - bal <= 100000000000000000000000 (LidCertifiedPresaleTimer.sol#46)
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - multiplier = bal.div(1000000000000000000000000).add(29) (LidCertifiedPresaleTimer.sol#47)
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - bal <= 100000000000000000000000 (LidCertifiedPresaleTimer.sol#48)
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - multiplier = bal.div(1000000000000000000000000).add(39) (LidCertifiedPresaleTimer.sol#49)
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - bal <= 1000000000000000000000000 (LidCertifiedPresaleTimer.sol#50)
LidCertifiedPresaleTimer.getEndTime(uint256) (LidCertifiedPresaleTimer.sol#38-58) uses literals with too many digits:
        - multiplier = bal.div(1000000000000000000000000).add(49) (LidCertifiedPresaleTimer.sol#51)
LidCertifiedPresale.sendToUniswap() (LidCertifiedPresale.sol#157-178) uses literals with too many digits:
        - uniswapRouter.addLiquidityETH.value(uniswapEth)(address(token),uniswapTokens,uniswapTokens,uniswapEth,address(0x000000000000000000000000000000000000dEaD)
,now) (LidCertifiedPresale.sol#170-177)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
INFO:Detectors:
ERC20Detailed._____gap (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#56) is never used in LidVotingRights (LidVotingRights.sol
#8-48)
Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61) is never used in LidTeamLock (LidTeamLock.sol#9-122)
Ownable._____gap (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#80) is never used in LidStakingV2 (LidStakingV2.sol#16-175)
Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61) is never used in LidPromoFund (LidPromoFund.sol#8-70)
Ownable._____gap (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#80) is never used in LidCertifiedPresaleTimer (LidCertifiedPresaleTimer
.sol#9-59)
Initializable._____gap (@openzeppelin/upgrades/contracts/Initializable.sol#61) is never used in LidStakingFund (LidStakingFund.sol#8-41)
ReentrancyGuard._____gap (@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol#41) is never used in LidCertifiedPresale (LidCertifiedPresa
le.sol#13-301)
LidCertifiedPresale.lidRepaired (LidCertifiedPresale.sol#59) is never used in LidCertifiedPresale (LidCertifiedPresale.sol#13-301)
Ownable._____gap (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#80) is never used in LidToken (LidToken.sol#16-178)
Ownable._____gap (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#80) is never used in LidDaoLock (LidDaoLock.sol#10-71)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
LidCertifiedPresale.lidRepaired (LidCertifiedPresale.sol#59) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
INFO:Detectors:
name() should be declared external:
        - LidToken.name() (LidToken.sol#116-118)
        - ERC20Detailed.name() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#28-30)
        - LidVotingRights.name() (LidVotingRights.sol#21-23)
symbol() should be declared external:
        - ERC20Detailed.symbol() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#36-38)
        - LidVotingRights.symbol() (LidVotingRights.sol#25-27)
decimals() should be declared external:
        - ERC20Detailed.decimals() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol#52-54)
        - LidVotingRights.decimals() (LidVotingRights.sol#29-31)
balanceOf(address) should be declared external:
        - LidVotingRights.balanceOf(address) (LidVotingRights.sol#33-35)
        - ERC20.balanceOf(address) (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#52-54)
totalSupply() should be declared external:
        - LidVotingRights.totalSupply() (LidVotingRights.sol#37-39)
        - ERC20.totalSupply() (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol#45-47)
balanceOfAt(address,uint256) should be declared external:
        - LidVotingRights.balanceOfAt(address,uint256) (LidVotingRights.sol#41-43)
totalSupplyAt(uint256) should be declared external:
        - LidVotingRights.totalSupplyAt(uint256) (LidVotingRights.sol#45-47)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol#19-21)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol#26-28)
totalStakedAt(uint256) should be declared external:
        - LidStakingV2.totalStakedAt(uint256) (LidStakingV2.sol#35-47)
stakeValueAt(address,uint256) should be declared external:
        - LidStakingV2.stakeValueAt(address,uint256) (LidStakingV2.sol#49-58)
addPauser(address) should be declared external:
        - PauserRole.addPauser(address) (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#31-33)
renouncePauser() should be declared external:
        - PauserRole.renouncePauser() (@openzeppelin/contracts-ethereum-package/contracts/access/roles/PauserRole.sol#35-37)
mint(address,uint256) should be declared external:
        - ERC20Mintable.mint(address,uint256) (@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Mintable.sol#25-28)
paused() should be declared external:
        - Pausable.paused() (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#43-45)
pause() should be declared external:
        - Pausable.pause() (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#66-69)
unpause() should be declared external:
        - Pausable.unpause() (@openzeppelin/contracts-ethereum-package/contracts/lifecycle/Pausable.sol#74-77)
addMinter(address) should be declared external:
        - MinterRole.addMinter(address) (@openzeppelin/contracts-ethereum-package/contracts/access/roles/MinterRole.sol#31-33)
renounceMinter() should be declared external:
        - MinterRole.renounceMinter() (@openzeppelin/contracts-ethereum-package/contracts/access/roles/MinterRole.sol#35-37)
registerAndStake(uint256) should be declared external:
        - LidStaking.registerAndStake(uint256) (LidStaking.sol#73-75)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
owner() should be declared external:
        - Ownable.owner() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#32-34)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#58-61)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol#67-69)
```

3.4 STATE VARIABLE VISIBILITY NOT SET - LOW

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Code Location:

LidDaoLock.sol: Line #23

LidCertifiedPresale.sol: Line #59, Line #60

LidStakingV2.sol: Line #23, Line #25

Recommendation:
Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables. Though all these state variables are meant to be internal, it is best practice to set the visibility.

3.5 NO RETURN VALUE - INFORMATIONAL

Description:
Defining a return type but the function is not explicitly returning any value.

Code Location:
LidPromoFund.sol: Line #37, Line #44, Line #49, Line #56
LidStakingFund.sol: Line #30, Line #37

Functions `releaseLidToAddress()`, `authorizeLid()`, `releaseEthToAddress()`, `authorizeEth()` has return type `uint` but they never explicitly returns any `uint` value.

Recommendation:
In `LidPromoFund` and `LidStakingFund` contract, `releaseLidToAddress()`, `authorizeLid()`, `releaseEthToAddress()`, `authorizeEth()` methods are external and has return type `uint`. It will fail the execution of external caller if caller is expecting return value from these methods. To remediate the issue, remove unnecessary return value type if method is not intended to return any value.

3.6 SECURITY FUZZING RESULT - INFORMATIONAL

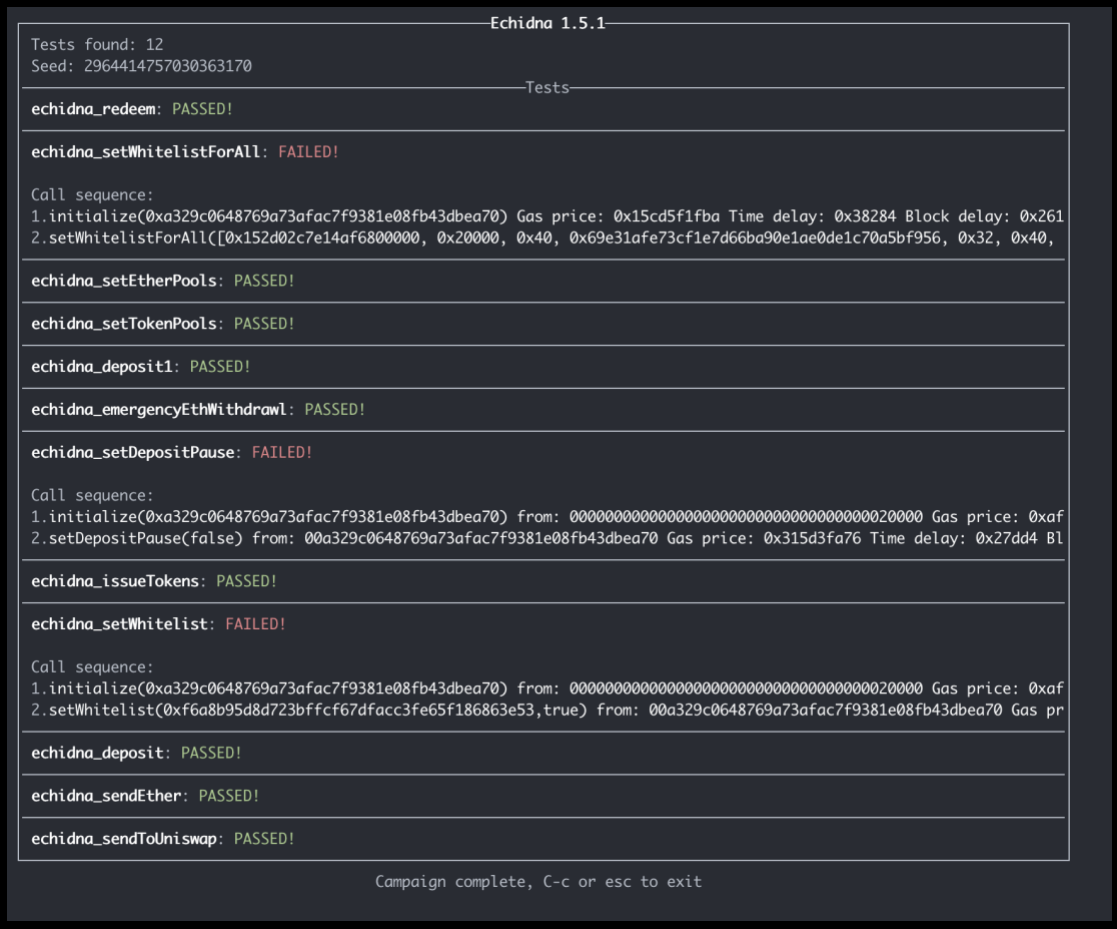
Description:

Fuzzing tests were performed on all the LID contracts having functions that are responsible to change the state of the state variables or global variables. These tests were carried out by echidna, which takes as input to the contract a list of properties that should always remain true. Although many of the contracts does not contain any Boolean property(variable) in their state variables, some Boolean properties were manually added to perform the fuzzing in all of the contracts. Also, echidna blindly fuzz the contract with various input values to all the methods. Echidna first call `initialize()` method of the contract and then try to hit every other methods which are configured to fuzz. Fuzzer comes up with many false positive results because it is designed to check constraints violation.

Fuzzing result of all the 11 contracts are passed and echidna is not able to find any issue. This section contains one sample fuzzing result of contract `LidCertifiedPresale.sol`. This result contains false positive results as well showing as “Failed” in the image. All these failed results have been manually analyzed and then concluding it as a false positive. Results of all the 11 contracts cannot be put here because of its result size.

Results:

LidCertifiedPresale.sol


Echidna 1.5.1
Tests found: 12
Seed: 2964414757030363170
Tests
echidna_redeem: PASSED!
echidna_setWhitelistForAll: FAILED!
Call sequence:
1.initialize(0xa329c0648769a73afac7f9381e08fb43dbea70) Gas price: 0x15cd5f1fba Time delay: 0x38284 Block delay: 0x261
2.setWhitelistForAll([0x152d02c7e14af6800000, 0x20000, 0x40, 0x69e31afe73cf1e7d66ba90e1ae0de1c70a5bf956, 0x32, 0x40,
echidna_setEtherPools: PASSED!
echidna_setTokenPools: PASSED!
echidna_deposit1: PASSED!
echidna_emergencyEthWithdrawl: PASSED!
echidna_setDepositPause: FAILED!
Call sequence:
1.initialize(0xa329c0648769a73afac7f9381e08fb43dbea70) from: 0000000000000000000000000000000000020000 Gas price: 0xaf
2.setDepositPause(false) from: 00a329c0648769a73afac7f9381e08fb43dbea70 Gas price: 0x315d3fa76 Time delay: 0x27dd4 Bl
echidna_issueTokens: PASSED!
echidna_setWhitelist: FAILED!
Call sequence:
1.initialize(0xa329c0648769a73afac7f9381e08fb43dbea70) from: 0000000000000000000000000000000000020000 Gas price: 0xaf
2.setWhitelist(0xf6a8b95d8d723bffcf67dfacc3fe65f186863e53,true) from: 00a329c0648769a73afac7f9381e08fb43dbea70 Gas pr
echidna_deposit: PASSED!
echidna_sendEther: PASSED!
echidna_sendToUniswap: PASSED!
Campaign complete, C-c or esc to exit

THANK YOU FOR CHOOSING

// HALBORN