// HALBORN

LID Liftoff

Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: January 1-9, 2021
Visit: Halborn.com

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/22/2020	Nishit Majithia
0.2	Document Edits	12/26/2020	Nishit Majithia
1.0	Final Version	12/31/2020	Nishit Majithia

CONTACT	COMPANY	EMAIL
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Nishit Majithia	Halborn	Nishit.Majithia@halborn.com

1.1 INTRODUCTION

LID engaged Halborn to conduct a security assessment of their LIDLiftOff smart contracts beginning on January 1st, 2021 and ending January 9th 2021. The security assessment was scoped to the contract LiftoffSettings.sol, LiftoffRegistration.sol, LiftoffEngine.sol, LiftoffInsurance.sol 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.

$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 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

LID LiftOff smart contracts including LiftoffSettings.sol, LiftoffRegistration.sol, LiftoffEngine.sol, LiftoffInsurance.sol. Specific commit of contract: commit 4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	1	2

SECURITY ANALYSIS	RISK LEVEL
INTEGER OVERFLOW	Medium
ADDRESS CHECK MISSING	Low
USE OF BLOCK.TIMESTAMP	Low
IGNORE RETURN VALUES	Informational
UNNECESSARY IMPORTS	Informational
STATIC ANALYSIS	Informational

FINDINGS & TECH DETAILS

3.1 INTEGER OVERFLOW - MEDIUM

Description:

An overflow happens when an arithmetic operation reaches the maximum size of a type. For instance in the contract LiftOffSettings.sol‘s setXethBP() method, require() statement is summing up few uint256 values which may end up overflowing the integer. In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits – either larger than the maximum or lower than the minimum representable value

Code Location:

LiftOffSettings.sol: Line #210-211 (https://github.com/Lid-Protocol/liftoff-core-contracts/blob/4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b/contracts/LiftoffSettings.sol)

```
202     function setXethBP(
203         uint256 _baseFeeBP,
204         uint256 _ethBuyBP,
205         uint256 _projectDevBP,
206         uint256 _mainFeeBP,
207         uint256 _lidPoolBP
208     ) public override {
209         require(
210             _baseFeeBP + _ethBuyBP + _projectDevBP + _mainFeeBP + _lidPoolBP ==
211                 10000,
212             "Must allocate 100% of eth raised"
213         );
214         baseFee = _baseFeeBP;
215         ethBuyBP = _ethBuyBP;
216         projectDevBP = _projectDevBP;
217         mainFeeBP = _mainFeeBP;
218         lidPoolBP = _lidPoolBP;
```

It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system.

3.2 ADDRESS CHECK MISSING - LOW

Description:

Address validation check is missing in LiftOffSettings.sol contract's many set methods which setting address value for state variables. The value of _val in setLiftoffInsurance(), setLiftoffRegistration(), setLiftoffEngine(), setXEth(), setXLocker(), setUniswapRouter(), setLidTreasury() and setLidPoolManager() which are passed during initialization is being directly assigned which can be zero or empty or invalid.

Location:

LiftOffSettings.sol: Line #113, #123, #133, #143, #153, #163, #183 and #193 (https://github.com/Lid-Protocol/liftoff-core-contracts/blob/4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b/contracts/LiftoffSettings.sol)

```
202     function setXethBP(
203         uint256 _baseFeeBP,
204         uint256 _ethBuyBP,
205         uint256 _projectDevBP,
206         uint256 _mainFeeBP,
207         uint256 _lidPoolBP
208     ) public override {
209         require(
210             _baseFeeBP + _ethBuyBP + _projectDevBP + _mainFeeBP + _lidPoolBP ==
211                 10000,
212             "Must allocate 100% of eth raised"
213         );
214         baseFee = _baseFeeBP;
215         ethBuyBP = _ethBuyBP;
216         projectDevBP = _projectDevBP;
217         mainFeeBP = _mainFeeBP;
218         lidPoolBP = _lidPoolBP;
```

Recommendation:

Check that the address is not zero or invalid before storing it into any state variable.

3.3 USE OF BLOCK.TIMESTAMP - LOW

Description:

Block timestamps have historically been used for a variety of applications, such as entropy for random numbers, locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

block.timestamp or its alias **now** can be manipulated by miners if they have some incentive to do so. LiftOffEngine.sol,

LiftOffInsurance.sol and LiftOffRegistration.sol contracts uses now at many places in the contracts.

Location:

LiftOffRegistration.sol: Line #44, #48 (https://github.com/Lid-Protocol/liftoff-core-contracts/blob/4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b/contracts/LiftoffRegistration.sol)

```
43          require(
44              launchTime >= block.timestamp + minLaunchTime,
45              "Not allowed to launch before minLaunchTime"
46          );
47          require(
48              launchTime <= block.timestamp + maxLaunchTime,
49              "Not allowed to launch after maxLaunchTime"
50          );
51          require(
52              totalSupplyWad < (10**12) * (10**18),
```

LiftOffInsurance.sol: Line #101, #115, #150, #202 (https://github.com/Lid-Protocol/liftoff-core-contracts/blob/4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b/contracts/LiftoffInsurance.sol)

```
99          require(
100             !isInsuranceExhausted(
101                 now,
102                 tokenInsurance.startTime,
103                 liftoffSettings.getInsurancePeriod(),
104                 xEthValue,
105                 tokenInsurance.baseXEth,
106                 tokenInsurance.redeemedXEth.add(xEthValue),
107                 tokenInsurance.claimedXEth,
108                 tokenInsurance.isUnwound
109             ),
110             "Redeem request exceeds available insurance."
111         );
112
113         if (
114             //Still in the first period (1 week)
115             now <=
116             tokenInsurance.startTime.add(
117                 liftoffSettings.getInsurancePeriod()
118             ) &&
```

```
149         uint256 cycles =
150             now.sub(tokenInsurance.startTime).div(
151                 liftoffSettings.getInsurancePeriod()
152             );
153
```

```
200
201         tokenInsurances[_tokenSaleId] = TokenInsurance({
202             startTime: now,
203             totalIgnited: totalIgnited,
204             tokensPerEthWad: rewardSupply
205                 .mul(1 ether)
206                 .div(totalIgnited.subBP(liftoffSettings.getBaseFeeBP()))
207                 .add(1), //division error safety margin,
208             baseXEth: totalIgnited.sub(
209                 totalIgnited.mulBP(liftoffSettings.getEthBuyBP())
210             ),
211             baseTokenLidPool: IERC20(deployed).balanceOf(address(this)),
212             redeemedXEth: 0,
213             claimedXEth: 0,
214             claimedTokenLidPool: 0,
```

LiftOffEngine.sol: Line #103, #346, #362, #374(https://github.com/Lid-Protocol/liftoff-core-contracts/blob/4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b/contracts/LiftoffEngine.sol)

```
98          require(
99              msg.sender == liftoffSettings.getLiftoffRegistration(),
100             "Sender must be LiftoffRegistration"
101         );
102         require(_endTime > _startTime, "Must end after start");
103         require(_startTime > now, "Must start in the future");
104         require(_hardCap >= _softCap, "Hardcap must be at least softCap");
105         require(_softCap >= 10 ether, "Softcap must be at least 10 ether");
106         require(_totalSupply >= 1000 * (10**18), "TotalSupply must be at least 1000 tokens");
107         require(_totalSupply < (10**12) * (10**18), "TotalSupply must be less than 1 trillion tokens");
108
109         tokenId = totalTokenSales;
110
111         tokens[tokenId] = TokenSale({
```

```
345         if (
346             (now <= endTime && totalIgnited < hardCap) ||
347             totalIgnited < softCap ||
348             isSparked
349         ) {
350             return false;
351         } else {
352             return true;
353         }
354     }
355
356     function isIgniting(
357         uint256 startTime,
358         uint256 endTime,
359         uint256 totalIgnited,
360         uint256 hardCap
361     ) public view override returns (bool) {
362         if (now < startTime || now > endTime || totalIgnited >= hardCap) {
363             return false;
364         } else {
365             return true;
366         }
367     }
368
369     function isRefunding(
370         uint256 endTime,
371         uint256 softCap,
372         uint256 totalIgnited
373     ) public view override returns (bool) {
374         if (totalIgnited >= softCap || now <= endTime) {
375             return false;
376         } else {
377             return true;
```

Recommendation:

Avoid relying on block.timestamp

3.4 IGNORE RETURN VALUES - INFORMATIONAL

Description:

The return value of an external call is not stored in a local or state variable. In contract LiftOffEngine.sol, there are few instances where external methods are being called and return values are being ignored

Recommendation:

Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

Location:

LiftOffEngine.sol: Line #448, #449 (https://github.com/Lid-Protocol/liftoff-core-contracts/blob/4d60af29855a7ed900b06aaffa0b83ac7bc7bc7b/contracts/LiftoffEngine.sol)

```
439     function _insuranceRegistration(
440         TokenSale storage tokenSale,
441         uint256 _tokenSaleId,
442         uint256 _xEthBuy
443     ) internal {
444         IERC20 deployed = IERC20(tokenSale.deployed);
445         uint256 toInsurance =
446             deployed.balanceOf(address(this)).sub(tokenSale.rewardSupply);
447         address liftoffInsurance = liftoffSettings.getLiftoffInsurance();
448         deployed.transfer(liftoffInsurance, toInsurance);
449         IXEth(liftoffSettings.getXEth()).transfer(
450             liftoffInsurance,
451             tokenSale.totalIgnited.sub(_xEthBuy)
452         );
453
454         ILiftoffInsurance(liftoffInsurance).register(_tokenSaleId);
455     }
```

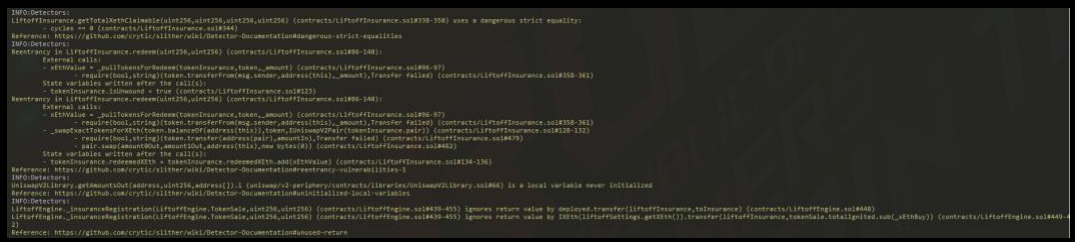
3.5 UNNECESSARY IMPORTS - INFORMATIONAL

Description:

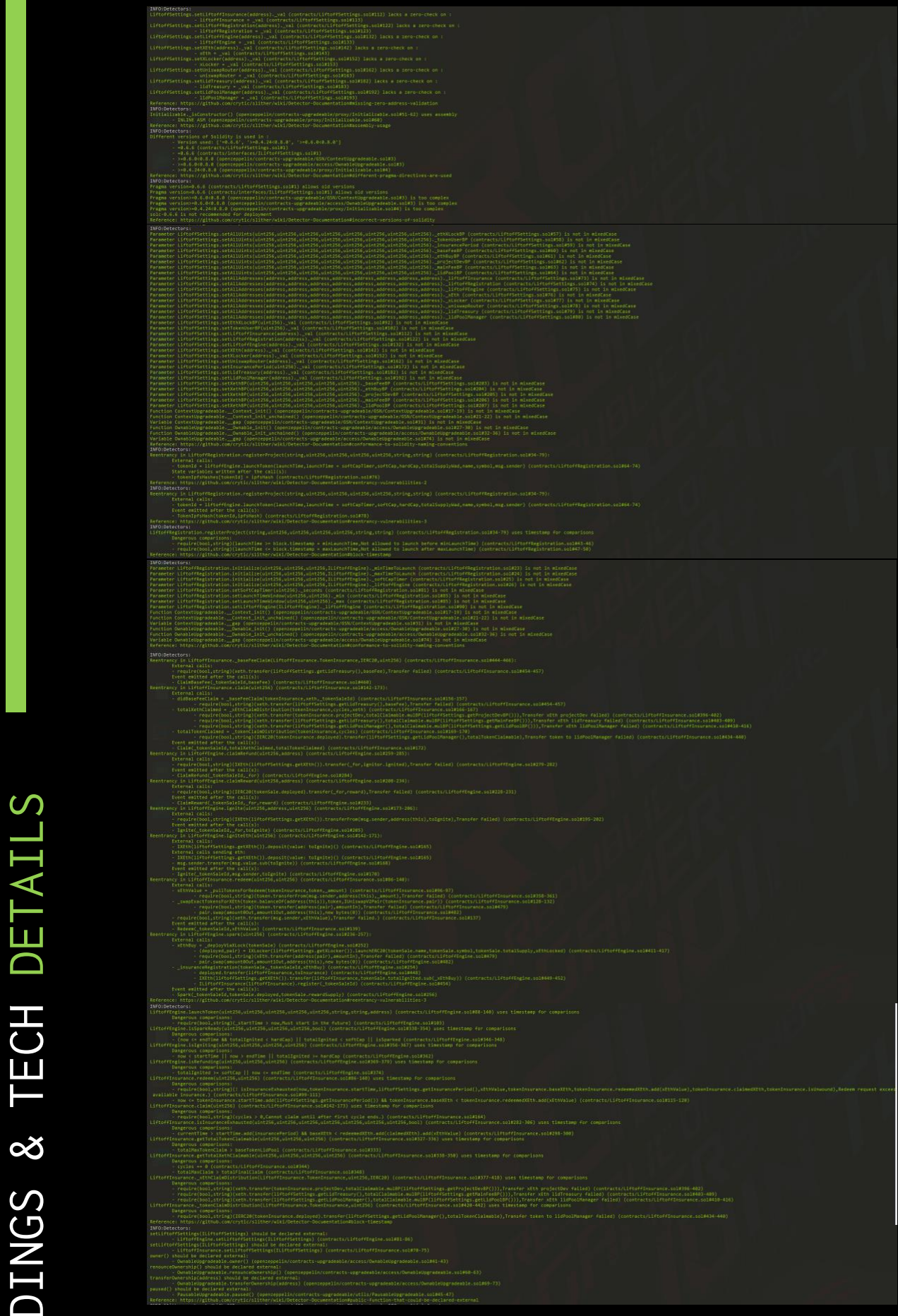
In contracts LiftOffEngine.sol and LiftOffInsuarance.sol, the Openzeppelin contract ReentrancyGuardUpgradeable.sol was being imported which was not required and was mitigated in commit f29cdf84605cff59b33db65865d1d4cae778e333 after bringing this to notice.

3.6 STATIC ANALYSIS -INFORMTIONAL

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-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.



FINDINGS & TECH DETAILS



MythX

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analysers to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with these scoped contracts

1. LiftOffEngine.sol

Report for contracts/LiftoffEngine.sol
https://dashboard.mythx.io/#/console/analyses/3bb2f77d-7288-4634-b6f7-df768968aacf

Line	SWC Title	Severity	Short Description
168	(SWC-134) Message call with hardcoded gas amount	Low	Call with hardcoded gas amount.
346	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
362	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
374	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

2. LiftOffRegistration.sol

Report for contracts/LiftoffRegistration.sol
https://dashboard.mythx.io/#/console/analyses/8ab5e0e8-93db-48c1-b133-b1861672bdff

Line	SWC Title	Severity	Short Description
18	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
22	(SWC-000) Unknown	Medium	Function could be marked as external.
43	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.
47	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

3. LiftOffSettings.sol

Report for contracts/LiftoffSettings.sol
https://dashboard.mythx.io/#/console/analyses/3e2cdddb-3205-488a-b2e4-ba5f15b29409

Line	SWC Title	Severity	Short Description
210	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operator can overflow.

THANK YOU FOR CHOOSING

// HALBORN