# Competitive Security Assessment

## Lymex

Nov 5th, 2022

Secure3

secure3.io

# Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

# Overview

### Project Detail

| | |
|---|---|
| **Project Name** | Lymex |
| **Platform & Language** | Solidity |
| **Codebase** | • audit code address - https://bscscan.com/address/0x78541049eb769AF9Cec448da6b581cb0BBB4D5f4#code<br>• final code address - https://bscscan.com/address/0xFE4Def3A81fCf69642b1CdDA86bDE72735F11EA3#code |
| **Audit Methodology** | • Audit Contest<br>• Business Logic and Code Review<br>• Privileged Roles Review<br>• Static Analysis |

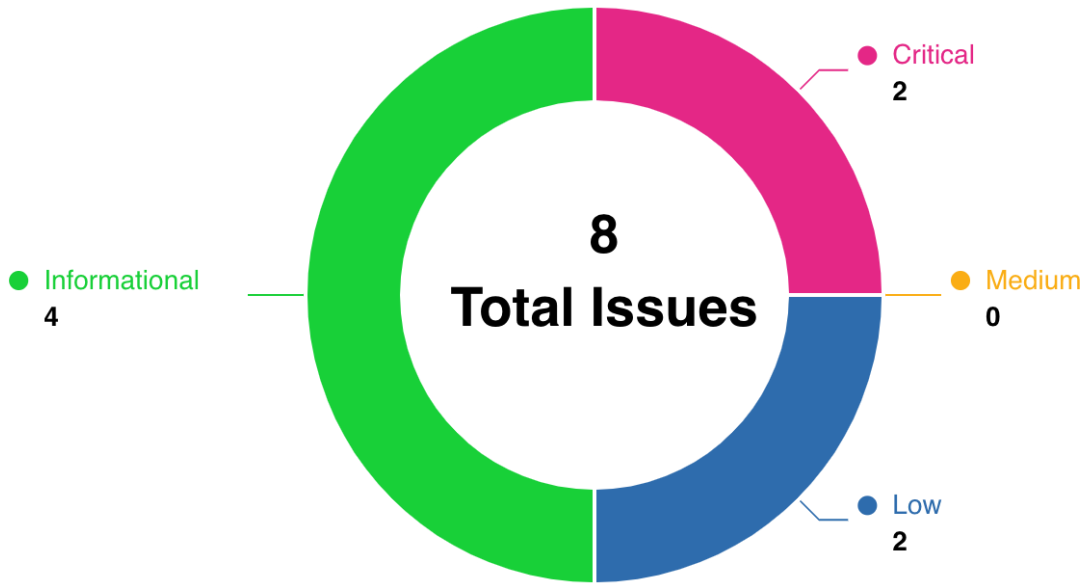
### Code Vulnerability Review Summary

| Vulnerability Level | Total | Reported | Acknowledged | Fixed | Mitigated | Declined |
|---|---|---|---|---|---|---|
| Critical | 2 | 0 | 2 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 | 0 | 0 |
| Low | 2 | 0 | 0 | 2 | 0 | 0 |
| Informational | 4 | 0 | 0 | 4 | 0 | 0 |

# Audit Scope

| File | Commit Hash |
| --- | --- |
| lym_sign_mining.sol | ca818ae8b510577f76f15008dafbb7019462fe61d4ab4cd2fac2b9d885de014e |
| divestor.sol | f5344cf89e8d4388a776b11bc5bb860b963fa77d5fae433d419f11399ce2c20a |

# Code Assessment Findings




| ID | Name | Category | Severity | Status | Contributor |
|---|---|---|---|---|---|
| LYM-1 | Inconsistent solidity compiler version | Language Specific | Informational | Fixed | Secure3 |
| LYM-2 | Risk of lost of fund due to zero address `payee_` in `Divestor` | Logical | Low | Fixed | Secure3 |
| LYM-3 | `registerSignMining` should use defined `onlyOpen` modifier | Code Style | Informational | Fixed | Secure3 |
| LYM-4 | Price manipulation risk in `LymSignPool::getPirce()` function | Oracle Manipulation | Critical | Acknowledged | Secure3 |
| LYM-5 | missing event in setter functions in `LymSignPool` | Code Style | Informational | Fixed | Secure3 |
| LYM-6 | Function `registerSignMining` should check if `nodeLevel_` is valid | Logical | Low | Fixed | Secure3 |
| LYM-7 | `swap` should set `amountOutMin` to avoid sandwich attack | Race condition | Critical | Acknowledged | Secure3 |
| LYM-8 | typo in `getPirce()` function name | Code Style | Informational | Fixed | Secure3 |

# LYM-1:Inconsistent solidity compiler version

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Language Specific | Informational | • code/contracts/other/divestor.sol#L2<br>• code/contracts/project/lym/lym_sign_mining.sol#L2 | Fixed | Secure3 |

## Code

```
2: pragma solidity =0.8.9;

2: pragma solidity ^0.8.4;
```

## Description

**Secure3 :** The compiler version of the two contracts are not the same, one `=0.8.9` and one `^0.8.4`

## Recommendation

**Secure3 :** Make the compiler version consistent in the project

## Client Response

Fixed. Changed the compiler version to be `pragma solidity =0.8.9;`

# LYM-2: Risk of lost of fund due to zero address `payee_` in `Divestor`

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Logical | Low | • code/contracts/other/divestor.sol#L19 | Fixed | Secure3 |

## Code

```
19:             payable(payee_).transfer(value_);
```

## Description

**Secure3 :** The passed in `payee_` address parameter can potentially be zero, when that is the case, the fund will be lost after transfer

## Recommendation

**Secure3 :** Check if `payee_` is zero address Consider below fix in the `sample.test()` function

```
require(payee_ != address(0), "payee_ is zero")
```

## Client Response

Fixed. Added the `require(payee_ != address(0), "payee_ is zero");` check

# LYM-3: `registerSignMining` should use defined `onlyOpen` modifier

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Code Style | Informational | • code/contracts/project/lym/lym_sign_mining.sol#L95 | Fixed | Secure3 |

## Code

```
95:         require(meta.isOpen, "not open");
```

## Description

**Secure3 :** use already defined `onlyOpen` modifier instead of coping the require check all the place

## Recommendation

**Secure3 :** function `registerSignMining` use `onlyOpen` modifier and remove `require(meta.isOpen, "not open");`

## Client Response

Fixed. Used `onlyOpen` modifier

# LYM-4:Price manipulation risk in `LymSignPool::getPirce()` function

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Oracle Manipulation | Critical | • code/contracts/project/lym/lym_sign_mining.sol#L226-232 | Acknowledged | Secure3 |

## Code

```
226:     function getPirce() public view returns (uint256) {
227:         (uint256 reserve0, uint256 reserve1, ) = addr.pair.getReserves();
228:         if (addr.pair.token0() == address(addr.usdt)) {
229:             return (reserve0 * 1e18) / reserve1;
230:         } else {
231:             return (reserve1 * 1e18) / reserve0;
232:         }
233:     }
```

## Description

**Secure3 :** the `getPirce()` function only gets the single tick price of that moment, which is very vulnerable to the price manipulation especially for the low liquidity tokens. this can lead to mis-compute `power` in the `registerSignMining()` function as `registerSignMining -> coutingPower -> getPirce`.

## Recommendation

**Secure3 :** use more robust algorithms such as time weighted average price (TWAP) to calculate the price.
More readings on

- https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles
- https://github.com/Uniswap/v2-periphery/blob/master/contracts/examples/ExampleSlidingWindowOracle.sol

## Client Response

Acknowledged. Contract already restricts the external contract to call the Lymex contract, hence the potential attack can only be achieved by calling the contract manually via EOA and the risk is reduced as the cost of manual operation will be too high for flash loan attack.

# LYM-5:missing event in setter functions in `LymSignPool`

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Code Style | Informational | • code/contracts/project/lym/lym_sign_mining.sol#L378-L411 | Fixed | Secure3 |

## Code

```
387:     function setFoundation(address foundation_) external onlyOwner {
388:         addr.foundation = foundation_;
389:     }
390:
391:     function setStaticRate(uint8 staticRate_) external onlyOwner {
392:         meta.staticRate = staticRate_;
393:     }
```

## Description

**Secure3 :** It is best practice to emit an event after the contract state is changed.

## Recommendation

**Secure3 :** add emit event statement at the end of the setter functions `setNodeInfo`, `setFoundation`, `setStaticRate`, `setReferRate`, `setOpen`,

## Client Response

Fixed. Added event for all the setter functions.

# LYM-6:Function `registerSignMining` should check if `nodeLevel_` is valid

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Logical | Low | • code/contracts/project/lym/lym_sign_mining.sol#L105 | Fixed | Secure3 |

## Code

```
105:         require(nodeLevel_ > user.node, "need to buy higher node");
```

## Description

**Secure3 :** The current `registerSignMining` only checks if `nodeLevel_` is higher than `user.node`. `nodeLevel_` may still be a node that has not been set and its needValue is `0`.

## Recommendation

**Secure3 :** Check if `nodeInfo[nodeLevel_]` is set.

```
require(node.needValue > 0, "invalid node");
```

## Client Response

Fixed. Added check for `node.needValue`.

# LYM-7: `swap` should set `amountOutMin` to avoid sandwich attack

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Race condition | Critical | • code/contracts/project/lym/lym_sign_mining.sol#L304 | Acknowledged | Secure3 |

## Code

```
    304:         addr.router.swapExactTokensForTokensSupportingFeeOnTransferTokens(amountUsdt_, 0,
path, address(this), block.timestamp + 120);
```

## Description

**Secure3 :** `swap` function swaps USDT for LYM tokens but dose not set `amountOutMin`. MEV bots can front-run `registerSignMining` transaction with a trade buying large amount of LYM tokens, and then sell tokens after the transaction to make user and protocol lose USDT.

## Recommendation

**Secure3 :** Get current LYM price off-chain and set a reasonable `amountOutMin` instead of `0`.

## Client Response

By design the user cannot control slippage.

# LYM-8:typo in `getPirce()` function name

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Code Style | Informational | • code/contracts/project/lym/lym_sign_mining.sol | Fixed | Secure3 |

## Code

```
226:     function getPirce() public view returns (uint256) {
```

## Description

**Secure3 :** typo in `lym_sign_mining` contract `getPirce()` function.

## Recommendation

**Secure3 :** fix the typo to `getPrice()`

## Client Response

Fixed. Fixed the typo.

# Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.