Lithium

# Smart Contract Audit Report

SYSFIXED

# TABLE OF CONTENTS

# AUDITED DETAILS

## Audited Project

| Project name | Token ticker | Blockchain |
|---|---|---|
| Lithium | LITH | Ethereum |

## Addresses

| Contract address | 0x188E817B02e635D482AE4D81e25DdA98A97C4a42 |
|---|---|
| Contract deployer address | 0x4aE6B6205ddEaff7f128Ad3F395859A30a650c18 |

## Project Website

| **http://lith.finance/** |
|---|

## Codebase

| **https://etherscan.io/address/0x188E817B02e635D482AE4D81e25DdA98A97C4a42#code** |
|---|

# SUMMARY

Lithium Finance is the first decentralized data oracle solution that uses collective intelligence in order to regularly price liquid assets which are difficult to evaluate.

## ▌Contract Summary

**Documentation Quality**

Lithium provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

**Code Quality**

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by Lithium with the discovery of several low issues.

**Test Coverage**

Test coverage of the project is 100% ( Through Codebase )

## ▌Audit Findings Summary

- SWC-101 | It is recommended to use vetted safe math libraries for arithmetic operations consistently on lines 138, 150, 163, 164, 175, 185, 199, 216, 231, 232, 250, 267, 289, 313 and 337.
- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 9, 34, 116, 343 and 670.

# CONCLUSION

We have audited the Lithium project released on January 2023 to discover issues and identify potential security vulnerabilities in Lithium Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides a satisfactory result with some low-risk issues.

The issues found in the Lithium smart contract code do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found are some arithmetic operation issues, and a floating pragma is set. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

# AUDIT RESULT

| Article | Category | Description | Result |
|---------|----------|-------------|--------|
| Default Visibility | SWC-100 SWC-108 | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | PASS |
| Integer Overflow and Underflow | SWC-101 | If unchecked math is used, all math operations should be safe from overflows and underflows. | ISSUE FOUND |
| Outdated Compiler Version | SWC-102 | It is recommended to use a recent version of the Solidity compiler. | PASS |
| Floating Pragma | SWC-103 | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | ISSUE FOUND |
| Unchecked Call Return Value | SWC-104 | The return value of a message call should be checked. | PASS |
| Unprotected Ether Withdrawal | SWC-105 | Due to missing or insufficient access controls, malicious parties can withdraw from the contract. | PASS |
| SELFDESTRUCT Instruction | SWC-106 | The contract should not be self-destructible while it has funds belonging to users. | PASS |
| Reentrancy | SWC-107 | Check effect interaction pattern should be followed if the code performs recursive call. | PASS |
| Uninitialized Storage Pointer | SWC-109 | Uninitialized local storage variables can point to unexpected storage locations in the contract. | PASS |
| Assert Violation | SWC-110 SWC-123 | Properly functioning code should never reach a failing assert statement. | PASS |
| Deprecated Solidity Functions | SWC-111 | Deprecated built-in functions should never be used. | PASS |
| Delegate call to Untrusted Callee | SWC-112 | Delegatecalls should only be allowed to trusted addresses. | PASS |

| | | | |
|---|---|---|---|
| DoS (Denial of Service) | SWC-113 SWC-128 | Execution of the code should never be blocked by a specific contract state unless required. | PASS |
| Race Conditions | SWC-114 | Race Conditions and Transactions Order Dependency should not be possible. | PASS |
| Authorization through tx.origin | SWC-115 | tx.origin should not be used for authorization. | PASS |
| Block values as a proxy for time | SWC-116 | Block numbers should not be used for time calculations. | PASS |
| Signature Unique ID | SWC-117 SWC-121 SWC-122 | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. | PASS |
| Incorrect Constructor Name | SWC-118 | Constructors are special functions that are called only once during the contract creation. | PASS |
| Shadowing State Variable | SWC-119 | State variables should not be shadowed. | PASS |
| Weak Sources of Randomness | SWC-120 | Random values should never be generated from Chain Attributes or be predictable. | PASS |
| Write to Arbitrary Storage Location | SWC-124 | The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations. | PASS |
| Incorrect Inheritance Order | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/. | PASS |
| Insufficient Gas Griefing | SWC-126 | Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract. | PASS |
| Arbitrary Jump Function | SWC-127 | As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value. | PASS |

| Typographical Error | SWC-129 | A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable. | PASS |
|---|---|---|---|
| Override control character | SWC-130 | Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract. | PASS |
| Unused variables | SWC-131 SWC-135 | Unused variables are allowed in Solidity and they do not pose a direct security issue. | PASS |
| Unexpected Ether balance | SWC-132 | Contracts can behave erroneously when they strictly assume a specific Ether balance. | PASS |
| Hash Collisions Variable | SWC-133 | Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision. | PASS |
| Hardcoded gas amount | SWC-134 | The transfer() and send() functions forward a fixed amount of 2300 gas. | PASS |
| Unencrypted Private Data | SWC-136 | It is a common misconception that private type variables cannot be read. | PASS |

# SMART CONTRACT ANALYSIS

| Started | Tuesday Jun 21 2022 11:09:29 GMT+0000 (Coordinated Universal Time) |
|---|---|
| Finished | Wednesday Jun 22 2022 05:16:33 GMT+0000 (Coordinated Universal Time) |
| Mode | Standard |
| Main Source File | LithiumToken.sol |

## Detected Issues

| ID | Title | Severity | Status |
|---|---|---|---|
| SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED | low | acknowledged |
| SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED | low | acknowledged |

| SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED | low | acknowledged |
|---------|-------------------------------------|-----|--------------|
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |

# SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED
LINE 138

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
137    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
138    uint256 c = a + b;
139    if (c < a) return (false, 0);
140    return (true, c);
141    }
142
```

# SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 150

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
149    if (b > a) return (false, 0);
150    return (true, a - b);
151    }
152
153    /**
154
```

# SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED
LINE 163

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
162   if (a == 0) return (true, 0);
163   uint256 c = a * b;
164   if (c / a != b) return (false, 0);
165   return (true, c);
166   }
167
```

# SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED

LINE 164

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
163   uint256 c = a * b;
164   if (c / a != b) return (false, 0);
165   return (true, c);
166   }
167
168
```

# SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED
LINE 175

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
174   if (b == 0) return (false, 0);
175   return (true, a / b);
176   }
177
178   /**
179
```

# SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED
LINE 185

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
184    if (b == 0) return (false, 0);
185    return (true, a % b);
186    }
187
188    /**
189
```

# SWC-101 | ARITHMETIC OPERATION "+" DISCOVERED
LINE 199

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
198    function add(uint256 a, uint256 b) internal pure returns (uint256) {
199    uint256 c = a + b;
200    require(c >= a, "SafeMath: addition overflow");
201    return c;
202    }
203
```

# SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 216

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
215    require(b <= a, "SafeMath: subtraction overflow");
216    return a - b;
217    }
218
219    /**
220
```

# SWC-101 | ARITHMETIC OPERATION "*" DISCOVERED
LINE 231

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
230    if (a == 0) return 0;
231    uint256 c = a * b;
232    require(c / a == b, "SafeMath: multiplication overflow");
233    return c;
234    }
235
```

# SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED
LINE 232

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
231   uint256 c = a * b;
232   require(c / a == b, "SafeMath: multiplication overflow");
233   return c;
234   }
235
236
```

# SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED
LINE 250

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
249    require(b > 0, "SafeMath: division by zero");
250    return a / b;
251    }
252
253    /**
254
```

# SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED
LINE 267

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
266    require(b > 0, "SafeMath: modulo by zero");
267    return a % b;
268    }
269
270    /**
271
```

# SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED
LINE 289

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
288    require(b <= a, errorMessage);
289    return a - b;
290    }
291
292    /**
293
```

# SWC-101 | ARITHMETIC OPERATION "/" DISCOVERED
LINE 313

## low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

## Source File

- LithiumToken.sol

## Locations

```
312   require(b > 0, errorMessage);
313   return a / b;
314   }
315
316   /**
317
```

# SWC-101 | ARITHMETIC OPERATION "%" DISCOVERED
LINE 337

## low SEVERITY
This plugin produces issues to support false positive discovery within mythril.

## Source File
- LithiumToken.sol

## Locations

```
336    require(b > 0, errorMessage);
337    return a % b;
338    }
339    }
340
341
```

# SWC-103 | A FLOATING PRAGMA IS SET.
LINE 9

## low SEVERITY

The current pragma Solidity directive is """>=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- LithiumToken.sol

## Locations

```
8
9    pragma solidity >=0.6.0 <0.8.0;
10
11   /*
12   * @dev Provides information about the current execution context, including the
13
```

# SWC-103 | A FLOATING PRAGMA IS SET.
LINE 34

## low SEVERITY

The current pragma Solidity directive is """>=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- LithiumToken.sol

## Locations

```
33
34    pragma solidity >=0.6.0 <0.8.0;
35
36    /**
37     * @dev Interface of the ERC20 standard as defined in the EIP.
38
```

# SWC-103 | A FLOATING PRAGMA IS SET.
LINE 116

## low SEVERITY

The current pragma Solidity directive is """>=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- LithiumToken.sol

## Locations

```
115
116    pragma solidity >=0.6.0 <0.8.0;
117
118    /**
119     * @dev Wrappers over Solidity's arithmetic operations with added overflow
120
```

# SWC-103 | A FLOATING PRAGMA IS SET.
LINE 343

## low SEVERITY

The current pragma Solidity directive is """>=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- LithiumToken.sol

## Locations

```
342
343    pragma solidity >=0.6.0 <0.8.0;
344
345    /**
346    * @dev Implementation of the {IERC20} interface.
347
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 670

## low SEVERITY

The current pragma Solidity directive is ""^0.7.6"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- LithiumToken.sol

## Locations

```
669
670    pragma solidity ^0.7.6;
671
672    /**
673    * @title LithiumToken
674
```

# 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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

# ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.