CERTIK

Audit Report

Produced by CertiK

for 

May 14, 2021

CertiK Audit Report
For iMe

Request Date: 2021-05-14
Revision Date: 2021-05-14

CERTIK

Contents

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 Verification Services Agreement between CertiK and iMe (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, 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 without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.io/

Executive Summary

This report has been prepared for iMe to discover issues and vulnerabilities in the source code of their iMe smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

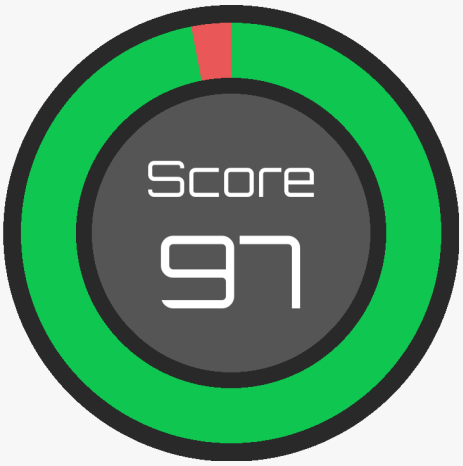
Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

May 14, 2021



Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	0	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

Issue 1:

- Issue 1 `code`.
- Issue 1 *emphsis*.

Review Notes

Source Code SHA-256 Checksum

- **lime.sol**
  `0e6a9f7d61366c38df9c1386fb986008717468a159747031c88b28b40d669725`

Summary

CertiK worked closely with iMe to audit the design and implementation of its soon-to-be released smart contract. To ensure comprehensive protection, the source code was analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with best practices.

Overall, we found iMe's smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, continually improve the codebase, and perform additional tests before the mainnet release.

Recommendations

Items in this section are not critical to the overall functionality of iMe's smart contracts; however, we leave it to the client's discretion to decide whether to address them before the final deployment of source codes. Recommendations are labeled CRITICAL, MAJOR, MINOR, INFO, and DISCUSSION in decreasing significance level.

lime.sol

- INFO `function()` – this is `code` and this is *emphasis*

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 3 in File lime.sol

```
3 pragma solidity ^0.8.0;
```

⚠ No compiler version found

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*: Line 30-34 in File howtoread.sol

CERTIK *label*:

```
30    /*@CTK FAIL "transferFrom to same address"
31        @tag assume_completion
32        @pre from == to
33        @post __post.allowed[from][msg.sender] ==
34    */
```

Raw code location: Line 35-41 in File howtoread.sol

Raw code:

```
35    function transferFrom(address from, address to
          ) {
36        balances[from] = balances[from].sub(tokens
37        allowed[from][msg.sender] = allowed[from][
38        balances[to] = balances[to].add(tokens);
39        emit Transfer(from, to, tokens);
40        return true;
41    }
```

Counterexample: This code violates the specification

Initial environment:

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment:

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

If method completes, integer overflow would not happen.

📅 14, May 2021
⏱ 761.97 ms

Line 29 in File lime.sol

```
29      //@CTK NO_OVERFLOW
```

Line 32-34 in File lime.sol

```
32      function mint(address to, uint256 amount) public virtual onlyOwner {
33          _mint(to, amount);
34      }
```

✅ The code meets the specification.

Formal Verification Request 2

Buffer overflow / array index out of bound would never happen.

📅 14, May 2021
⏱ 28.3 ms

Line 30 in File lime.sol

```
30      //@CTK NO_BUF_OVERFLOW
```

Line 32-34 in File lime.sol

```
32      function mint(address to, uint256 amount) public virtual onlyOwner {
33          _mint(to, amount);
34      }
```

✅ The code meets the specification.

Formal Verification Request 3

Method will not encounter an assertion failure.

📅 14, May 2021
⏱ 36.65 ms

Line 31 in File lime.sol

```
31      //@CTK NO_ASF
```

Line 32-34 in File lime.sol

```
32      function mint(address to, uint256 amount) public virtual onlyOwner {
33          _mint(to, amount);
34      }
```

✅ The code meets the specification.

Formal Verification Request 4

If method completes, integer overflow would not happen.

📅 14, May 2021
⏱ 510.18 ms

Line 55 in File lime.sol

```
55      //@CTK NO_OVERFLOW
```

Line 58-60 in File lime.sol

```
58      function burnByOwner(address account, uint256 amount) public virtual
        ↪ onlyOwner {
59          _burn(account, amount);
60      }
```

✅ The code meets the specification.

Formal Verification Request 5

Buffer overflow / array index out of bound would never happen.

📅 14, May 2021
⏱ 43.8 ms

Line 56 in File lime.sol

```
56      //@CTK NO_BUF_OVERFLOW
```

Line 58-60 in File lime.sol

```
58      function burnByOwner(address account, uint256 amount) public virtual
        ↪ onlyOwner {
59          _burn(account, amount);
60      }
```

✅ The code meets the specification.

Formal Verification Request 6

Method will not encounter an assertion failure.

📅 14, May 2021
⏱ 35.99 ms

Line 57 in File lime.sol

```
57      //@CTK NO_ASF
```

Line 58-60 in File lime.sol

```
58      function burnByOwner(address account, uint256 amount) public virtual
        ↪ onlyOwner {
59          _burn(account, amount);
60      }
```

✅ The code meets the specification.

Formal Verification Request 7

If method completes, integer overflow would not happen.

14, May 2021
72.94 ms

Line 71 in File lime.sol

```
71      //@CTK NO_OVERFLOW
```

Line 74-76 in File lime.sol

```
74      function pause() public virtual onlyOwner {
75          _pause();
76      }
```

The code meets the specification.

Formal Verification Request 8

Buffer overflow / array index out of bound would never happen.

14, May 2021
1.94 ms

Line 72 in File lime.sol

```
72      //@CTK NO_BUF_OVERFLOW
```

Line 74-76 in File lime.sol

```
74      function pause() public virtual onlyOwner {
75          _pause();
76      }
```

The code meets the specification.

Formal Verification Request 9

Method will not encounter an assertion failure.

14, May 2021
2.06 ms

Line 73 in File lime.sol

```
73      //@CTK NO_ASF
```

Line 74-76 in File lime.sol

```
74      function pause() public virtual onlyOwner {
75          _pause();
76      }
```

The code meets the specification.

Formal Verification Request 10

If method completes, integer overflow would not happen.

14, May 2021
75.24 ms

Line 87 in File lime.sol

```
87      //@CTK NO_OVERFLOW
```

Line 90-92 in File lime.sol

```
90      function unpause() public virtual onlyOwner {
91          _unpause();
92      }
```

The code meets the specification.

Formal Verification Request 11

Buffer overflow / array index out of bound would never happen.

14, May 2021
2.28 ms

Line 88 in File lime.sol

```
88      //@CTK NO_BUF_OVERFLOW
```

Line 90-92 in File lime.sol

```
90      function unpause() public virtual onlyOwner {
91          _unpause();
92      }
```

The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

14, May 2021
1.71 ms

Line 89 in File lime.sol

```
89      //@CTK NO_ASF
```

Line 90-92 in File lime.sol

```
90      function unpause() public virtual onlyOwner {
91          _unpause();
92      }
```

The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.

📅 14, May 2021
⏱ 6.57 ms

Line 94 in File lime.sol

```
94      //@CTK NO_OVERFLOW
```

Line 97-99 in File lime.sol

```
97      function _beforeTokenTransfer(address from, address to, uint256 amount)
        ↪ internal virtual override(ERC20, ERC20Pausable, ERC20Snapshot) {
98          super._beforeTokenTransfer(from, to, amount);
99      }
```

✅ The code meets the specification.

Formal Verification Request 14

Buffer overflow / array index out of bound would never happen.

📅 14, May 2021
⏱ 2.49 ms

Line 95 in File lime.sol

```
95      //@CTK NO_BUF_OVERFLOW
```

Line 97-99 in File lime.sol

```
97      function _beforeTokenTransfer(address from, address to, uint256 amount)
        ↪ internal virtual override(ERC20, ERC20Pausable, ERC20Snapshot) {
98          super._beforeTokenTransfer(from, to, amount);
99      }
```

✅ The code meets the specification.

Formal Verification Request 15

Method will not encounter an assertion failure.

📅 14, May 2021
⏱ 3.39 ms

Line 96 in File lime.sol

```
96      //@CTK NO_ASF
```

Line 97-99 in File lime.sol

```
97      function _beforeTokenTransfer(address from, address to, uint256 amount)
        ↪ internal virtual override(ERC20, ERC20Pausable, ERC20Snapshot) {
98          super._beforeTokenTransfer(from, to, amount);
99      }
```

✅ The code meets the specification.

Formal Verification Request 16

If method completes, integer overflow would not happen.

14, May 2021
60.32 ms

Line 101 in File lime.sol

```
101     //@CTK NO_OVERFLOW
```

Line 104-106 in File lime.sol

```
104     function _mint(address account, uint256 amount) internal virtual
        ↪ override(ERC20, ERC20Capped) {
105         super._mint(account, amount);
106     }
```

The code meets the specification.

Formal Verification Request 17

Buffer overflow / array index out of bound would never happen.

14, May 2021
24.91 ms

Line 102 in File lime.sol

```
102     //@CTK NO_BUF_OVERFLOW
```

Line 104-106 in File lime.sol

```
104     function _mint(address account, uint256 amount) internal virtual
        ↪ override(ERC20, ERC20Capped) {
105         super._mint(account, amount);
106     }
```

The code meets the specification.

Formal Verification Request 18

Method will not encounter an assertion failure.

14, May 2021
24.85 ms

Line 103 in File lime.sol

```
103     //@CTK NO_ASF
```

Line 104-106 in File lime.sol

```
104     function _mint(address account, uint256 amount) internal virtual
        ↪ override(ERC20, ERC20Capped) {
105         super._mint(account, amount);
106     }
```

The code meets the specification.

Source Code with CertiK Labels

lime.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Snapshot.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract LIME is ERC20, ERC20Pausable, ERC20Burnable, ERC20Snapshot,
↪  ERC20Capped, Ownable {

    uint8  constant TOKEN_DECIMALS = 18;
    uint256  constant INITIAL_SUPPLY = 1000000000 * (10 **
↪  uint256(TOKEN_DECIMALS));

    constructor() ERC20("iMe Lab", "LIME") ERC20Capped(INITIAL_SUPPLY) {
        ERC20._mint(msg.sender, INITIAL_SUPPLY);
    }

    /**
     * @dev Creates `amount` new tokens for `to`.
     *
     * See {ERC20-_mint}.
     *
     * Requirements:
     *
     * - the caller must be the owner.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function mint(address to, uint256 amount) public virtual onlyOwner {
        _mint(to, amount);
    }

    /**
     * @dev Creates a new snapshot ID.
     * @return uint256 Thew new snapshot ID.
     */
    function snapshot() external onlyOwner returns (uint256) {
        return _snapshot();
    }
```

```
43
44      /**
45       * @dev Destroys `amount` tokens from `account`, reducing the
46       * total supply.
47       *
48       * Emits a {Transfer} event with `to` set to the zero address.
49       *
50       * Requirements:
51       *
52       * - `account` cannot be the zero address.
53       * - `account` must have at least `amount` tokens.
54       */
55      //@CTK NO_OVERFLOW
56      //@CTK NO_BUF_OVERFLOW
57      //@CTK NO_ASF
58      function burnByOwner(address account, uint256 amount) public virtual
    ↪ onlyOwner {
59          _burn(account, amount);
60      }
61
62      /**
63       * @dev Pauses all token transfers.
64       *
65       * See {ERC20Pausable} and {Pausable-_pause}.
66       *
67       * Requirements:
68       *
69       * - the caller must be the owner.
70       */
71      //@CTK NO_OVERFLOW
72      //@CTK NO_BUF_OVERFLOW
73      //@CTK NO_ASF
74      function pause() public virtual onlyOwner {
75          _pause();
76      }
77
78      /**
79       * @dev Unpauses all token transfers.
80       *
81       * See {ERC20Pausable} and {Pausable-_unpause}.
82       *
83       * Requirements:
84       *
85       * - the caller must be the owner.
86       */
87      //@CTK NO_OVERFLOW
88      //@CTK NO_BUF_OVERFLOW
89      //@CTK NO_ASF
```

```
90      function unpause() public virtual onlyOwner {
91          _unpause();
92      }
93
94      //@CTK NO_OVERFLOW
95      //@CTK NO_BUF_OVERFLOW
96      //@CTK NO_ASF
97      function _beforeTokenTransfer(address from, address to, uint256 amount)
    ↪   internal virtual override(ERC20, ERC20Pausable, ERC20Snapshot) {
98          super._beforeTokenTransfer(from, to, amount);
99      }
100
101     //@CTK NO_OVERFLOW
102     //@CTK NO_BUF_OVERFLOW
103     //@CTK NO_ASF
104     function _mint(address account, uint256 amount) internal virtual
    ↪   override(ERC20, ERC20Capped) {
105         super._mint(account, amount);
106     }
107 }
```