# TEAM FINANCE LOCK SMART CONTRACT AUDIT

# FOR DAO TEAM Finance

**13.10.2020**

**Made in Germany by Chainsulting.de**

# Smart Contract Audit

# 1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Team Finance. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

| Major Versions / Date | Description |
| --- | --- |
| 0.1 (12.10.2020) | Layout |
| 0.5 (13.10.2020) | Automated Security Testing<br>Manual Security Testing |
| 0.8 (13.10.2020) | Adding of MythX and SWC |
| 1.0 (13.10.2020) | Summary and Recommendation |
| 2.0 (14.10.2020) | Final document |

## 2. About the Project and Company

**Company address: N/A (DAO)**

**Voting Governance: N/A**

**Team: N/A**

**Website:** https://team.finance

**GitHub: N/A**

**Twitter:** https://twitter.com/teamfinance_

**Discord:** https://discord.com/invite/yCCj4N3

**Youtube:** https://www.youtube.com/channel/UCjMZfolatlfZ1LxnP1z7wfQ

**Telegram:** https://t.me/team_finance

**Medium:** https://medium.com/@teamfinance_

## 2.1 Project Overview

Team Finance is a decentralized platform allowing token creators and developers to create a “smart contract lock” to hold their ERC20 and Uniswap liquidity tokens for a specified period of time. This creates trust in the token community and prevents spams and “rug pulls”.

## 3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

| Level | Value | Vulnerability | Risk (Required Action) |
|---|---|---|---|
| Critical | 9 – 10 | A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken. | Immediate action to reduce risk level. |
| High | 7 – 8.9 | A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way. | Implementation of corrective actions as soon as possible. |
| Medium | 4 – 6.9 | A vulnerability that could affect the desired outcome of executing the contract in a specific scenario. | Implementation of corrective actions in a certain period. |
| Low | 2 – 3.9 | A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective. | Implementation of certain corrective actions or accepting the risk. |
| Informational | 0 – 1.9 | A vulnerability that have informational character but is not effecting any of the code. | An observation that does not determine a level of risk |

# 4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

## 4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

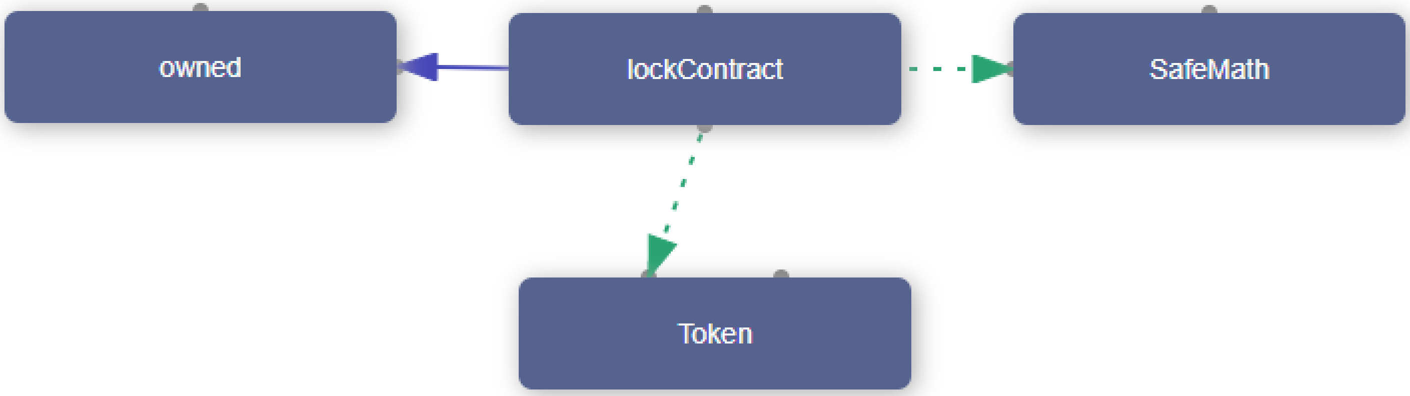
## 4.3 Tested Contract Files

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

| File | Fingerprint (SHA256) | Source |
|---|---|---|
| team_finance_lockContract.sol | 4c014a11769bbb853317db21df5576a02427f0ff4981fa17350450763b1cdd8f | https://raw.githubusercontent.com/chainsulting/Smart-Contract-Security-Audits/master/Team%20Finance/team_finance_lockContract.sol |

# 5. Summary of Smart Contract

## 5.1 Visualized Dependencies



## 5.2 Functions

| contract | func | visibility | modifiers | stateMutability |
|---|---|---|---|---|
| Token | balanceOf | external | | view |
| Token | allowance | external | | view |
| Token | transfer | external | | |
| Token | approve | external | | |
| Token | approveAndCall | external | | |
| Token | transferFrom | external | | |
| SafeMath | mul | internal | | constant |
| SafeMath | div | internal | | constant |
| SafeMath | sub | internal | | constant |
| SafeMath | add | internal | | constant |
| SafeMath | ceil | internal | | constant |
| owned | owned | public | | |
| owned | transferOwnership | public | onlyOwner | |
| lockContract | lockContract | public | | |
| lockContract | lockTokens | public | | |
| lockContract | transferLocks | public | | |
| lockContract | withdrawTokens | public | | |
| lockContract | getTotalTokenBalance | public | | view |
| lockContract | getTokenBalanceByAddress | public | | view |
| lockContract | getAllDepositIds | public | | view |
| lockContract | getDepositDetails | public | | view |
| lockContract | numOfActiveDeposits | public | | view |

| | | | | |
|---|---|---|---|---|
| lockContract | getWithdrawableDepositsByAddress | public | | view |
| lockContract | getAllDepositsByAddress | public | | view |

## 5.3 Modifiers

| contract | modifier |
|---|---|
| owned | onlyOwner |

## 5.4 States

| contract | state | type | visibility | isConst |
|---|---|---|---|---|
| owned | owner | address | public | false |
| lockContract | depositId | uint256 | public | false |
| lockContract | allDepositIds | array | public | false |
| lockContract | depositsByWithdrawalAddress | mapping | public | false |
| lockContract | lockedToken | mapping | public | false |
| lockContract | walletTokenBalance | mapping | public | false |

# 6. Test Suite Results

The Lock Contract is part of the Team Finance platform and this one was audited. All the functions and state variables are well commented using inline documentation for the functions which is good to understand quickly how everything is supposed to work.

## 6.1 Mythril Classic & MYTHX Automated Vulnerability Test

Mythril Classic is an open-source security analysis tool for Ethereum smart contracts. It uses concolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities. Also now known as MYTHX and part of Consensys AG

### 6.1.1 A floating pragma is set

Severity: LOW
Code: SWC-103
File(s) affected: team_finance_lockContract.sol

| Attack / Description | Code Snippet | Result/Recommendation |
|---|---|---|
| The current pragma Solidity directive is `^0.4.16;`<br>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. | Line: 1<br>`pragma solidity ^0.4.16;` | It is recommended to follow the example (0.4.16), as future compiler versions may handle certain language constructions in a way the developer did not foresee. Not effecting the overall contract functionality. |

6.1.2 A control flow decision is made based on The block.timestamp environment variable

Severity: LOW

Code: SWC-116

File(s) affected: team_finance_lockContract.sol

| Attack / Description | Code Snippet | Result/Recommendation |
|---|---|---|
| The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners. | Line: 137<br>`require(block.timestamp >= lockedToken[_id].unlockTime);` | Not used for randomness and not effecting the overall contract functionality. |

### 6.1.3 Implicit loop over unbounded data structure

Severity: LOW
Code: SWC-128
File(s) affected: team_finance_lockContract.sol

| Attack / Description | Code Snippet | Result/Recommendation |
|---|---|---|
| Gas consumption in function "getAllDepositIds" in contract "lockContract" depends on the size of data structures that may grow unboundedly. The highlighted statement involves copying the array "allDepositIds" from "storage" to "memory". When copying arrays from "storage" to "memory" the Solidity compiler emits an implicit loop.If the array grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose. | Line: 176 - 180<br>`function numOfActiveDeposits(address _withdrawalAddress) public view returns (uint256) {`<br>`uint256 staked = 0;`<br>`for (uint i = 0; i < depositsByWithdrawalAddress[_withdrawalAddress].length; i++) {`<br>`if (!lockedToken[depositsByWithdrawalAddress[_withdrawalAddress][i]].withdrawn) {`<br>`staked++;` | Unlikely by the use case to result in large gas cost |

## 6.2. Slither, Oyente, Solhint, HoneyBadger Automated Vulnerability Test

No more issues were identified.

## 6.3. SWC Attacks & Manual Security Testing

| ID | Title | Relationships | Test Result |
|---|---|---|---|
| SWC-131 | Presence of unused variables | CWE-1164: Irrelevant Code | ✅ |
| SWC-130 | Right-To-Left-Override control character (U+202E) | CWE-451: User Interface (UI) Misrepresentation of Critical Information | ✅ |
| SWC-129 | Typographical Error | CWE-480: Use of Incorrect Operator | ✅ |
| SWC-128 | DoS With Block Gas Limit | CWE-400: Uncontrolled Resource Consumption | ❌ |
| SWC-127 | Arbitrary Jump with Function Type Variable | CWE-695: Use of Low-Level Functionality | ✅ |
| SWC-125 | Incorrect Inheritance Order | CWE-696: Incorrect Behavior Order | ✅ |

| ID | Title | Relationships | Test Result |
|---|---|---|---|
| SWC-124 | Write to Arbitrary Storage Location | CWE-123: Write-what-where Condition | ✅ |
| SWC-123 | Requirement Violation | CWE-573: Improper Following of Specification by Caller | ✅ |
| SWC-122 | Lack of Proper Signature Verification | CWE-345: Insufficient Verification of Data Authenticity | ✅ |
| SWC-121 | Missing Protection against Signature Replay Attacks | CWE-347: Improper Verification of Cryptographic Signature | ✅ |
| SWC-120 | Weak Sources of Randomness from Chain Attributes | CWE-330: Use of Insufficiently Random Values | ✅ |
| SWC-119 | Shadowing State Variables | CWE-710: Improper Adherence to Coding Standards | ✅ |
| SWC-118 | Incorrect Constructor Name | CWE-665: Improper Initialization | ✅ |
| SWC-117 | Signature Malleability | CWE-347: Improper Verification of Cryptographic Signature | ✅ |
| SWC-116 | Timestamp Dependence | CWE-829: Inclusion of Functionality from Untrusted Control Sphere | ❌ |

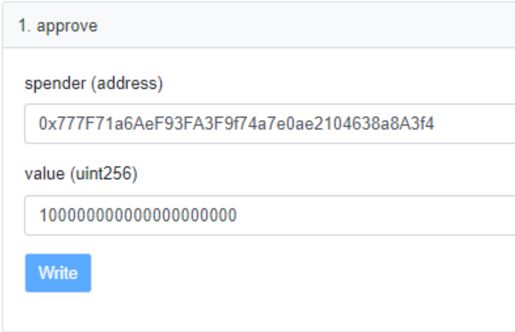
| ID | Title | Relationships | Test Result |
|---|---|---|---|
| SWC-115 | Authorization through tx.origin | CWE-477: Use of Obsolete Function | ✅ |
| SWC-114 | Transaction Order Dependence | CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') | ✅ |
| SWC-113 | DoS with Failed Call | CWE-703: Improper Check or Handling of Exceptional Conditions | ✅ |
| SWC-112 | Delegatecall to Untrusted Callee | CWE-829: Inclusion of Functionality from Untrusted Control Sphere | ✅ |
| SWC-111 | Use of Deprecated Solidity Functions | CWE-477: Use of Obsolete Function | ✅ |
| SWC-110 | Assert Violation | CWE-670: Always-Incorrect Control Flow Implementation | ✅ |
| SWC-109 | Uninitialized Storage Pointer | CWE-824: Access of Uninitialized Pointer | ✅ |
| SWC-108 | State Variable Default Visibility | CWE-710: Improper Adherence to Coding Standards | ✅ |
| SWC-107 | Reentrancy | CWE-841: Improper Enforcement of Behavioral Workflow | ✅ |
| SWC-106 | Unprotected SELFDESTRUCT Instruction | CWE-284: Improper Access Control | ✅ |

| ID | Title | Relationships | Test Result |
|---|---|---|---|
| SWC-105 | Unprotected Ether Withdrawal | CWE-284: Improper Access Control | ✅ |
| SWC-104 | Unchecked Call Return Value | CWE-252: Unchecked Return Value | ✅ |
| SWC-103 | Floating Pragma | CWE-664: Improper Control of a Resource Through its Lifetime | ❌ |
| SWC-102 | Outdated Compiler Version | CWE-937: Using Components with Known Vulnerabilities | ✅ |
| SWC-101 | Integer Overflow and Underflow | CWE-682: Incorrect Calculation | ✅ |
| SWC-100 | Function Default Visibility | CWE-710: Improper Adherence to Coding Standards | ✅ |

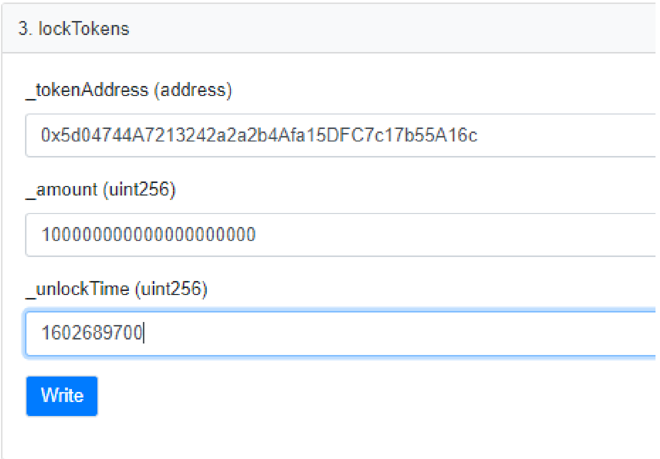
Sources:
https://smartcontractsecurity.github.io/SWC-registry
https://dasp.co
https://github.com/chainsulting/Smart-Contract-Security-Audits
https://consensys.github.io/smart-contract-best-practices/known_attacks

Testing deployment
https://ropsten.etherscan.io/address/0x777f71a6aef93fa3f9f74a7e0ae2104638a8a3f4#code

1.) Use the approve function of the token that will be locked and fill the spender address with the locking contract
https://ropsten.etherscan.io/tx/0x1bdb0fe68261757e6e517b800b113d00a0f1fd44a113bc838143ed58bebadf63




2.) Lock of a token amount and specific unlock time (https://www.unixtimestamp.com/index.php)
https://ropsten.etherscan.io/tx/0xf67c06febb60036b1ca6653612c84946a93420a2d51d7a998de5c87b5db308b4



3.) Check if deposit was tracked (Read)

5. getDepositDetails

_id (uint256)

3

Query

∟ tokenAddress *address*, withdrawalAddress *address*, tokenAmount *uint256*, unlockTime *uint256*, withdrawn *bool*

[ **getDepositDetails** method Response ]

» **tokenAddress** *address* : 0x5d04744A7213242a2a2b4Afa15DFC7c17b55A16c

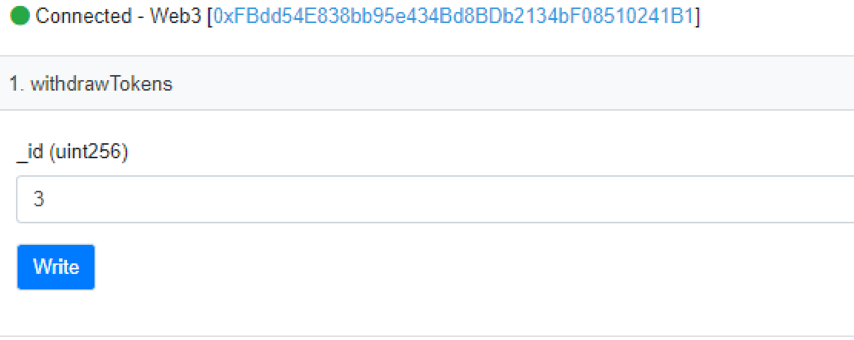
» **withdrawalAddress** *address* : 0x7705ad243e362650b0bD78a54F8Fd06Af11a89c9

» **tokenAmount** *uint256* : 100000000000000000000

» **unlockTime** *uint256* : 1602689700

» **withdrawn** *bool* : false

4.) Trying to withdraw token after locking time end, with contract deployer address and deposit address (Rug Pull by Team Finance)

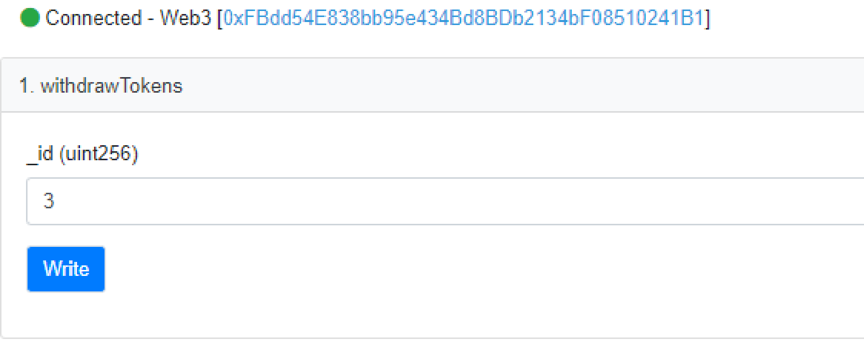



Result:
Failed by any other address or team.finance deployer:
https://ropsten.etherscan.io/tx/0x43c8a996a37422263a394e72713f0a7947e7f2091d06096273de99338a98ae1e

Success by locking address:
https://ropsten.etherscan.io/tx/0x827951751db38b7c6da54e2e5a90597eb580cf9946b81c531a77b165131735f6

Means it's successful possible by the token owner to withdraw the amount after lock period ends and not possible for someone else. It is working as expected

5.) Trying to withdraw token before locking time end, with contract deployer address and deposit address (Rug Pull by Team Finance)




Result:
Failed by any other address or team.finance deployer:
https://ropsten.etherscan.io/tx/0xc2bd0260b60e15c3a9b60cc18087703e9d9d74d4c537551dbe35fcf78559770c

Failed by locking address:
https://ropsten.etherscan.io/tx/0xde3fc9b27430c36928ebbb493967f24795b99b8bfcd42827a7e6c09d4183a71d

# 7. Executive Summary

The smart contract is written as simple as possible and also not overloaded with unnecessary functions, these is greatly benefiting the security of the contract. It correctly implemented widely-used and reviewed contracts for safe mathematical operations. The audit identified no major security vulnerabilities, at the moment of audit. We noted that a majority of the functions were self-explanatory, and inline documentation were included. No critical issues were found after the manual and automated security testing.

# 8. Deployed Smart Contract

**Lock Contract**

https://etherscan.io/address/0xdbf72370021babafbceb05ab10f99ad275c6220a#code