SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Trade Leaf

Audit

Security Assessment

07. March, 2023

For

Tradeleaf

Disclaimer 3
Description 5
Project Engagement 5
Logo 5
Contract Link 5
Methodology 7
Used Code from other Frameworks/Smart Contracts (direct imports) 8
Tested Contract Files 9
Source Lines 10
Risk Level 10
Capabilities 11
Inheritance Graph 12
CallGraph 13
Scope of Work/Verify Claims 14
Modifiers and public functions 24
Source Units in Scope 26
Critical issues 27
High issues 27
Medium issues 27
Low issues 27
Informational issues 27
Audit Comments 27
SWC Attacks 28

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing 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. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	25. February 2023	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	7. March 2023	• Reaudit

Network
Polygon

Website
https://tradeleaf.io/

Telegram
https://t.me/tradeleaforg

Twitter
https://twitter.com/tradeleaf

LinkedIN
https://www.linkedin.com/company/tradeleaf

Description

Trade Leaf is a technology company driven by a team of traditional trade finance business leaders, blockchain visionaries, venture investors, and tech entrepreneurs cooperating to disrupt the conservative and bureaucratized trade finance industry.
We create, integrate, and provide institutional-grade FinTech solutions and services for all trade finance industry participants. Our goal is to cover the trillion market gap supporting SMEs and MSMEs on emerging market.

Project Engagement

During the Date of 25 February 2023, **Trade Leaf Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

- Provided as Files

v1.1

- https://github.com/TradeLeaf/TLF-token-smartcontracts/tree/main/contracts
- Commit: 41441763e3f33e092cbe6becdb9f59a118e3a893

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 aspossible.
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

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.

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 SolidProof 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 SolidProof 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.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

```
@openzeppelin/contracts/security/ReentrancyGuard.sol
@openzeppelin/contracts/access/AccessControl.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
```

```
@openzeppelin/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol
@openzeppelin/contracts/security/Pausable.sol
@openzeppelin/contracts/access/Ownable.sol
```

```
@openzeppelin/contracts/access/AccessControl.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol
./ITLFVesting.sol
```

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

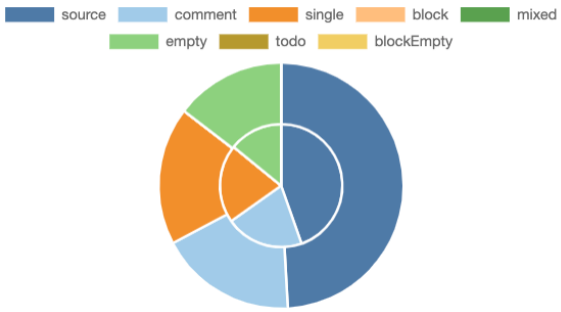
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

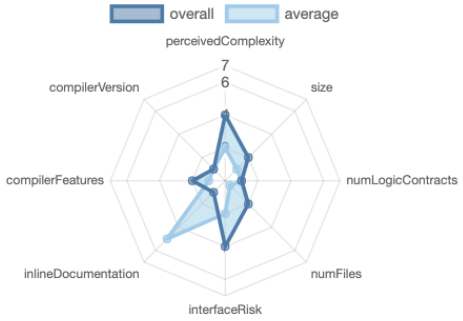
File Name	SHA-1 Hash
contracts/ ITLFLaunchpad.sol	48dab0be92f8ba36c6e818f231baf82c8ca7e314
contracts/TLFVesting.sol	e126b35586bfdff9d825d036ff80787e6e445674
contracts/ TLFLaunchpad.sol	da8224cf882abec23d20270663a4cbe4757002b8
contracts/TLFToken.sol	8307c4b526a866db6b7c789b66527771d0d373a2
contracts/ITLFVesting.sol	019a2984ae1c53be0ad03075bfd803878f1dc8e4

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	💊Abstract
3	0	2	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
55	0

External	Internal	Private	Pure	View
51	34	3	1	28

StateVariables

Total	🌐Public
12	2

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`^0.8.18`				

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪚 ECRecover	🌀 New/Create/Create2
			`yes`		

♻️ TryCatch	Σ Unchecked

Inheritance Graph
v1.0

CallGraph
v1.0

Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Is contract an upgradeable
2. Correct implementation of Token standard
3. Deployer cannot mint any new tokens
4. Deployer cannot burn or lock user funds
5. Deployer cannot pause the contract
6. Deployer cannot set fees
7. Deployer cannot blacklist/antisnipe addresses
8. Overall checkup (Smart Contract Security)

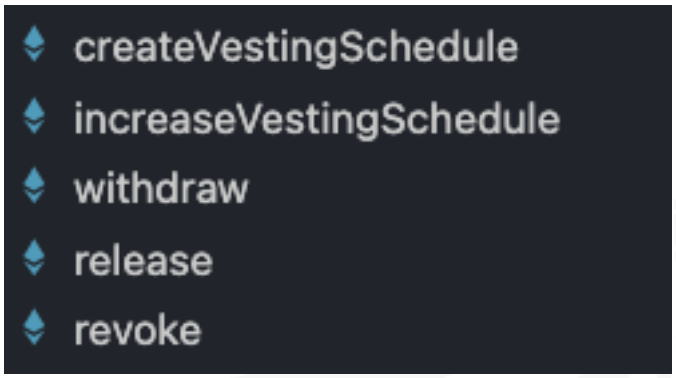
Is contract an upgradeable

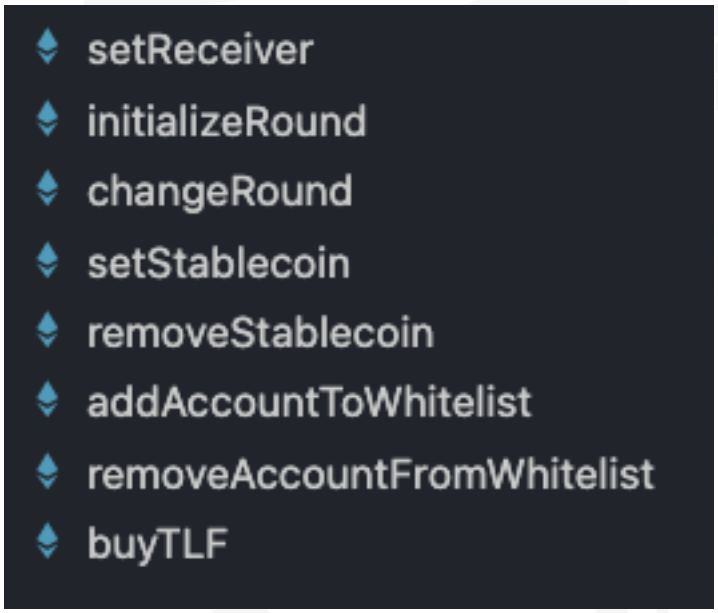
Name	
Is contract an upgradeable?	No

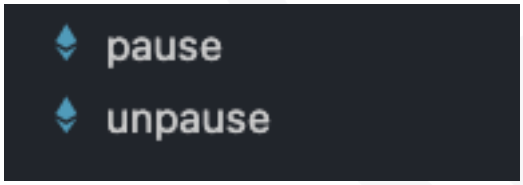
Correct implementation of Token standard

ERC20				
Function	**Description**	**Exist**	**Tested**	**Verified**
TotalSupply	Provides information about the total token supply	✓	✓	✓
BalanceOf	Provides account balance of the owner's account	✓	✓	✓
Transfer	Executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	Executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	Allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	Returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract v1.0







Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	3.000.000.000		

Comments:

v1.0

- Owner cannot mint new tokens

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer can lock	✓	✓	✗
Deployer cannot burn	-	-	-

Comments:

v1.0

- Owner can lock the transfer of funds for the users by pausing the token contract

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer can pause	✓	✓	✗

Comments:

v1.0

- Owner can pause the token contract

Deployer cannot set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 25%	–	–	–
Deployer cannot set fees to nearly 100% or to 100%	–	–	–

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer can blacklist/antisnipe addresses	✓	✓	✗

Comments:

v1.0

- Only whitelist addresses will be able to buy TLF and the owner can remove wallets from the whitelist

Overall checkup (Smart Contract Security)

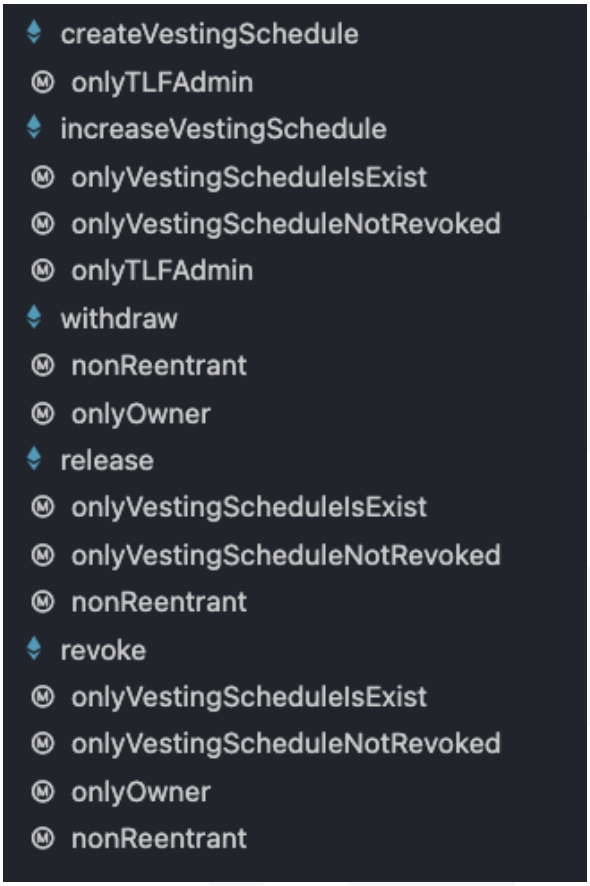
Tested	Verified
✓	✓

Legend

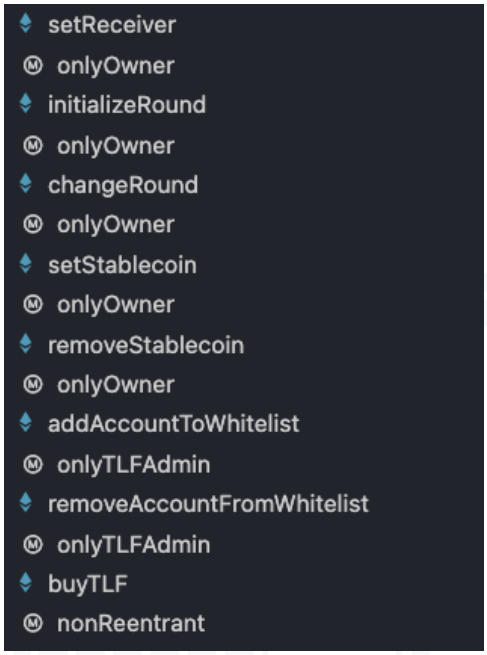
Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	-

Modifiers and public functions
v1.0

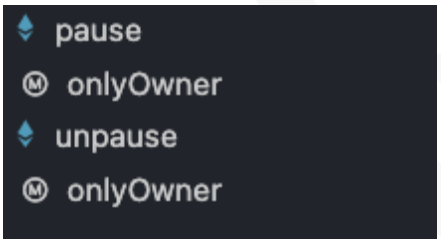
TLFVesting.sol



TLFLaunchpad



TLFToken.sol



Ownership Privileges:

- *TLFVesting:*
  - The admins address/wallet can create a new vesting schedule, and set duration, amount, to any arbitrary value.
  - Increase the vesting schedule but only
  - The owner can withdraw funds from the contract but the amount left after subtracting the vested token balance from the contracts balance
  - Revoke vesting schedule if it was set as revocable at the time of vesting creation.
  - Admin is able to release to an address with vesting schedule

- *TLFLaunchpad:*
  - The owner can set stable coin receiver address and change it at anytime
  - Initialise round and set startTime, duration, total amount, limit, price, vesting lock, and duration to any arbitrary value.
  - Change the initialised round, change the duration but not after the round is over, but decreasing/increasing of the duration is possible
  - It is also possible to change the total amount, and it must be more than sold amount
  - Owner can also changes the price
  - The tokens bought from the launchpad contract will vested and not be received instantly in the caller address.

- There are several authorities which are authorized to call some functions, that means, if the owner is renounced, another address is still authorized to call functions
  - Be aware of this

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope
v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/ITLFLaunchpad.sol	—	1	107	61	48	1	29
contracts/TLFVesting.sol	1	—	318	282	142	93	99
contracts/TLFLaunchpad.sol	1	—	373	351	182	104	121
contracts/TLFToken.sol	1	—	29	29	22	1	24
contracts/ITLFVesting.sol	—	1	76	46	38	1	25
Totals	**3**	**2**	**903**	**769**	**432**	**200**	**298**

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	All	Missing Events Arithmetic	All	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	Main	NatSpec documentation missing	—	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/latest/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

7. March 2023:

- There is still an owner (Owner still has not renounced ownership)
- Read whole report and modifiers section for more information

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

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

SolidProof_io

@solidproof_io

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY