



**TrueLayer**

**Stablecoin**

**SMART CONTRACT AUDIT**

**05.10.2022**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer.....	4
2. About the Project and Company .....	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level .....	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology .....	8
5. Metrics .....	9
5.1 Tested Contract Files .....	9
5.2 Used Code from other Frameworks/Smart Contracts .....	10
5.3 CallGraph .....	11
5.4 Inheritance Graph .....	12
5.5 Source Lines & Risk .....	13
5.6 Capabilities .....	14
5.7 Source Unites in Scope .....	15
6. Scope of Work.....	16
6.1 Findings Overview .....	17
6.2 Manual and Automated Vulnerability Test.....	18
6.2.1 Privileged Roles.....	18
6.2.2 Public Functions Could Be External.....	19
6.2.3 Missing Value Verification.....	20
6.2.4 Floating Pragma Version Identified .....	20
6.2.5 Missing Natspec Documentation .....	21

6.3 SWC Attacks .....	22
6.4 Verify Claims .....	26
6.5 Unit Tests .....	27
6.5.1 Unit Tests Coverage .....	28
7. Executive Summary .....	29
8. Mainnet Deployment .....	29
9. About the Auditor .....	30



## 1. Disclaimer

The audit makes no statements or warranties 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 TrueLayer Limited. 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 (21.09.2022)	Layout
0.4 (22.09.2022)	Automated Security Testing Manual Security Testing
0.5 (24.09.2022)	Verify Claims and Test Deployment
0.6 (25.09.2022)	Testing SWC Checks
0.9 (25.09.2022)	Summary and Recommendation
1.0 (26.09.2022)	Final document
1.1 (05.10.2022)	Re-check
1.2 (TBA)	Deployed contract

## 2. About the Project and Company

**Company address:**

TrueLayer Limited  
1 Hardwick Street  
London, EC1R 4RB  
UK



**Website:** <https://truelayer.com>

**Twitter:** <https://twitter.com/TrueLayer>

**Facebook:** <https://www.facebook.com/TrueLayerAPI>

**GitHub:** <https://github.com/TrueLayer>

**LinkedIn:** <https://www.linkedin.com/company/truelayer>

**YouTube:** <https://www.youtube.com/channel/UCgCJvUI1ViLmINzwPoRjcTw>

## 2.1 Project Overview

TrueLayer is a fintech platform utilized to build financial apps that connect to bank data, verify accounts, and access transactions.

TrueLayer is a global open banking platform. Businesses use the TrueLayer open banking network to securely access financial data and enable instant payments.

More than half of the open banking traffic in the UK, Ireland and Spain goes through TrueLayer and payments convert 20% higher than other open banking providers. TrueLayer was one of the first to be regulated in the UK to provide: account information services (AIS) and payment initiation services (PIS). These open banking services allow consumers to view their accounts in one place, or pay from their bank without having to enter credit card details. TrueLayer's APIs are fully compliant with UK and European Open Banking standards.

TrueLayer also provides electronic money services to its merchant clients and acts as a technical service provider or intermediary for regulated open banking providers. They can also help merchants who are not regulated yet to access open banking through an agent model.

### 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.



## 5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

### 5.1 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 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 (MD5)
./contracts/Ethereum/Blacklistable.sol	d78ba320f867f35819de480986a00686
./contracts/Ethereum/TIStablecoinV010.sol	23bef07bbce84e6a74a1b11eddef5794

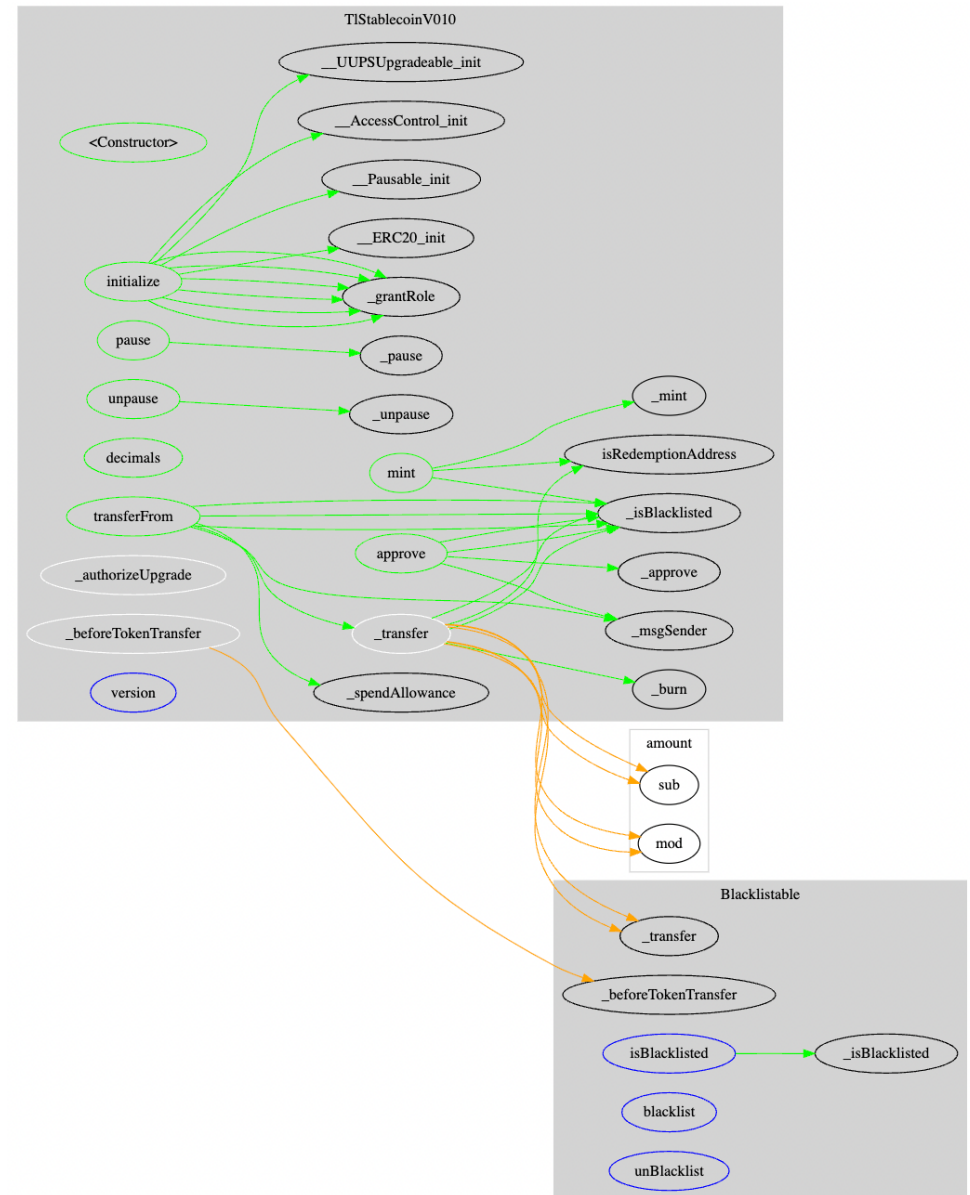
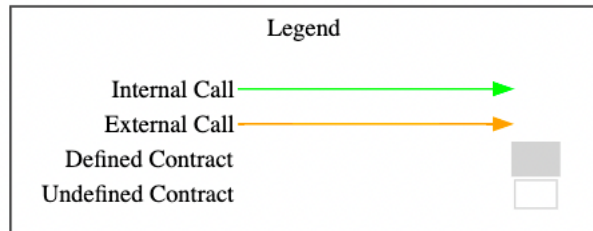
Updated (05.10.2022)

File	Fingerprint (MD5)
./contracts/Ethereum/Blacklistable.sol	36f33c01a3f3db75cdde9d0e6daaca0f
./contracts/Ethereum/TIStablecoinV010.sol	67a67c4e8c3dd1b06008a845e12f4191

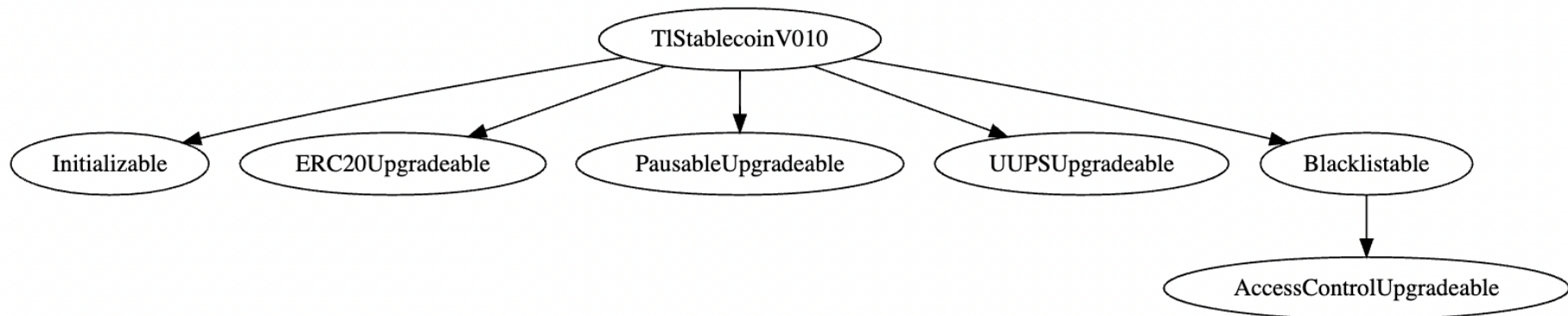
## 5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/access/AccessControlUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/access/AccessControlUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/proxy/utils/Initializable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/proxy/utils/Initializable.sol</a>
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/proxy/utils/UUPSUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/proxy/utils/UUPSUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/security/PausableUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/security/PausableUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/token/ERC20/ERC20Upgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/token/ERC20/ERC20Upgradeable.sol</a>
@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/utils/math/SafeMathUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.7.3/utils/math/SafeMathUpgradeable.sol</a>

## 5.3 CallGraph

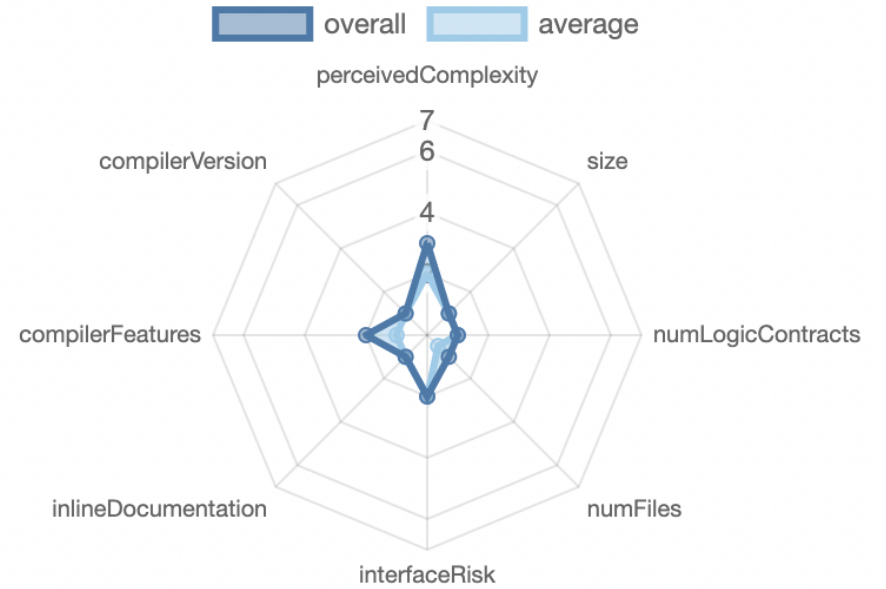
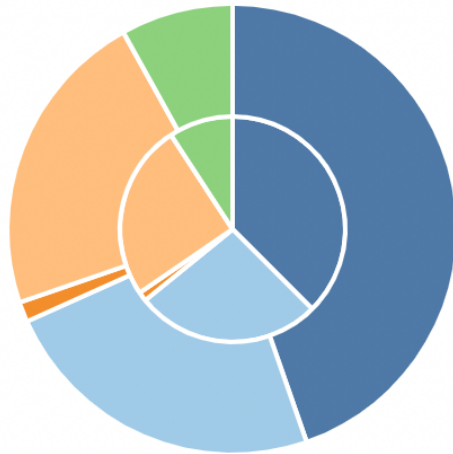


## 5.4 Inheritance Graph













## 5.5 Source Lines & Risk

source comment single block mixed  
empty todo blockEmpty





## 5.6 Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<input type="text" value="^0.8.4"/>			<input type="text"/>	<input type="text"/>	<input type="text"/>
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTrecover	 New/Create/Create2
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="yes"/>	<input type="text"/>	<input type="text"/>

### 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				
<input type="text" value="11"/>	<input type="text" value="0"/>				
External	Internal	Private	Pure	View	
<input type="text" value="4"/>	<input type="text" value="17"/>	<input type="text" value="0"/>	<input type="text" value="2"/>	<input type="text" value="3"/>	

### StateVariables

Total	 Public
<input type="text" value="9"/>	<input type="text" value="7"/>

## 5.7 Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/Ethereum/TIStablecoinV010.sol	1		197	161	93	49	106	
	contracts/Ethereum/Blacklistable.sol	1		68	62	21	33	17	
	<b>Totals</b>	<b>2</b>		<b>265</b>	<b>223</b>	<b>114</b>	<b>82</b>	<b>123</b>	

Legend: [ ]

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized 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, ...)

## 6. Scope of Work

The TrueLayer team provided us with the files that needs to be tested. The scope of the audit is the stablecoin contract.

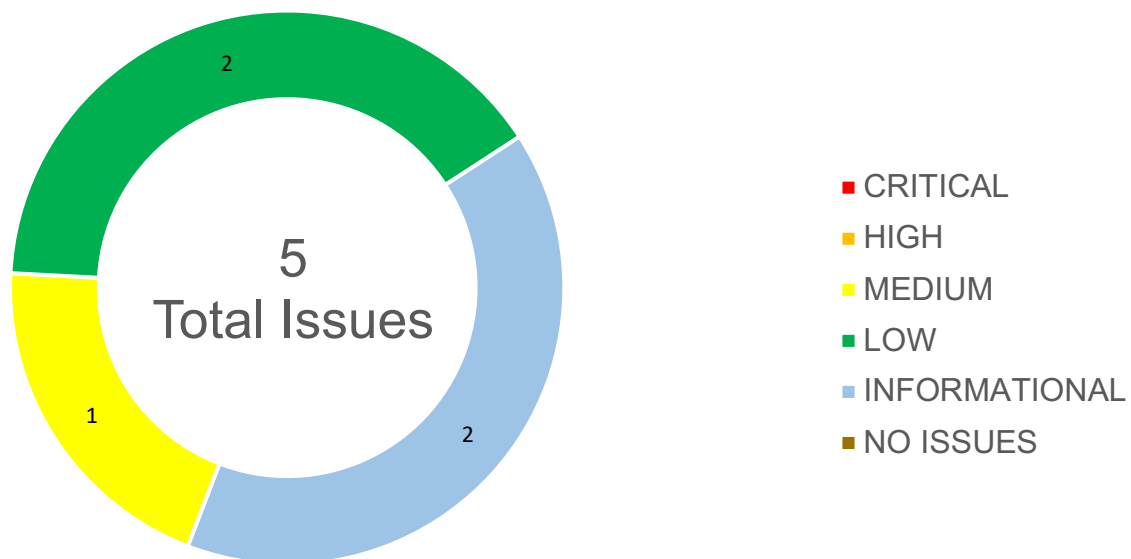
The team put forward the following assumptions regarding the security, usage of the contracts:

- The ERC-20 Upgradable Token standard is correctly implemented
- Roles are correctly implemented
- Blacklisting is working as expected
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



## 6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Privileged Roles	MEDIUM	CLOSED
6.2.2	Public Functions Could Be External	LOW	CLOSED
6.2.3	Missing Value Verification	LOW	CLOSED
6.2.4	Floating Pragma Version Identified	INFORMATIONAL	CLOSED
6.2.5	Missing Natspec Documentation	INFORMATIONAL	CLOSED

## 6.2 Manual and Automated Vulnerability Test

### CRITICAL ISSUES

During the audit, Chainsulting's experts found **0 Critical issues** in the code of the smart contract.

### HIGH ISSUES

During the audit, Chainsulting's experts found **0 High issues** in the code of the smart contract.

### MEDIUM ISSUES

During the audit, Chainsulting's experts found **1 Medium issue** in the code of the smart contract.

#### 6.2.1 Privileged Roles

Severity: MEDIUM

Status: CLOSED

Code: NA

File(s) affected: TIStablecoinV010.sol

Update: TrueLayer commits to guidelines and mechanisms aimed at minimising all potential risks related to it. This is done by implementing technical and procedural controls for all the roles associated with the stablecoin.

<b>Attack / Description</b>	The ROLES have the privilege to execute pause, mint, blacklist and upgrade. This represents a significant centralization risk where the ROLES have privileged control over the contract. The auditor has not recognized any multi-signature structure.
<b>Code</b>	Line 20 - 25 (TIStablecoinV010.sol)

	<pre> bytes32 public constant UPGRADER_ROLE = keccak256('UPGRADER_ROLE'); bytes32 public constant PAUSER_ROLE = keccak256('PAUSER_ROLE'); bytes32 public constant UNPAUSER_ROLE = keccak256('UNPAUSER_ROLE'); bytes32 public constant MINTER_ROLE = keccak256('MINTER_ROLE'); bytes32 public constant BLACKLISTER_ROLE = keccak256('BLACKLISTER_ROLE'); bytes32 public constant UNBLACKLISTER_ROLE = keccak256('UNBLACKLISTER_ROLE'); </pre>
<b>Result/Recommendation</b>	As the ROLES and centralized control is required by regulations for stablecoins, the wallets associated with ROLES should be secured by multi-signature.

## LOW ISSUES

During the audit, Chainsulting's experts found **2 Low issues** in the code of the smart contract.

### 6.2.2 Public Functions Could Be External

Severity: LOW

Status: CLOSED

Code: NA

File(s) affected: TIStablecoinV010.sol

Update: Fixed within commit 563386a0e4cf26f2d37be34c6741253c7fa0774f

<b>Attack / Description</b>	In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.
<b>Code</b>	Line 43 initialize(...) Line 57 pause() Line 61 unpause()

	Line 66 mint(...)
<b>Result/Recommendation</b>	We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

### 6.2.3 Missing Value Verification

Severity: LOW

Status: CLOSED

Code: NA

File(s) affected: TIStablecoinV010.sol

Update: Fixed within commit 563386a0e4cf26f2d37be34c6741253c7fa0774f

<b>Attack / Description</b>	The initialize functions lack a value safety check. Therefore, only values that are consistent with the logic of the contract should be permitted. Missing address validation checks could lead to contract deployment without properly set admin, upgrader, pauser or blacklister roles.
<b>Code</b>	Line 34 – 55 (TIStablecoinV010.sol) initialize(...)
<b>Result/Recommendation</b>	It is recommended to check address values for correctness. This can be done in first stage to exclude zero address in a require statement. In second stage to check if an address is a contract and most specific for contracts if an address implements a specified interface (EIP-165).

## INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **2 Informational issues** in the code of the smart contract.

### 6.2.4 Floating Pragma Version Identified

Severity: INFORMATIONAL

Status: CLOSED

Code: SWC-103

File(s) affected: ALL

Update: Fixed within commit 563386a0e4cf26f2d37be34c6741253c7fa0774f

<b>Attack / Description</b>	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.
<b>Code</b>	e.g. Line 2 <code>pragma solidity ^0.8.4;</code>
<b>Result/Recommendation</b>	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.  i.e. <code>pragma solidity 0.8.4</code>

## 6.2.5 Missing Natspec Documentation

Severity: INFORMATIONAL

Status: CLOSED

Code: NA

File(s) affected: ALL

Update: Fixed within commit 563386a0e4cf26f2d37be34c6741253c7fa0774f

<b>Attack / Description</b>	Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).
<b>Code</b>	NA

<b>Result/Recommendation</b>	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract.
------------------------------	--

## 6.3 SWC Attacks

ID	Title	Relationships	Test Result
<a href="#">SWC-131</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	✓
<a href="#">SWC-130</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	✓
<a href="#">SWC-129</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	✓
<a href="#">SWC-128</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	✓
<a href="#">SWC-127</a>	Arbitrary Jump with Function Type Variable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	✓
<a href="#">SWC-125</a>	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	✓
<a href="#">SWC-124</a>	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-123</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	✓
<a href="#">SWC-122</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	✓
<a href="#">SWC-121</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-120</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	✓
<a href="#">SWC-119</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-118</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	✓
<a href="#">SWC-117</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-116</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-115</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	✓


ID	Title	Relationships	Test Result
<a href="#">SWC-114</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	✓
<a href="#">SWC-113</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	✓
<a href="#">SWC-112</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-111</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-110</a>	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	✓
<a href="#">SWC-109</a>	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	✓
<a href="#">SWC-108</a>	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-107</a>	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	✓
<a href="#">SWC-106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	✓




ID	Title	Relationships	Test Result
<a href="#">SWC-104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	✓
<a href="#">SWC-103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	✓
<a href="#">SWC-102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	✓
<a href="#">SWC-101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	✓
<a href="#">SWC-100</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓

## 6.4 Verify Claims


6.4.1 The ERC-20 Upgradable Token standard is correctly implemented

**Status:** tested and verified 


6.4.2 Roles are correctly implemented

**Status:** tested and verified 

6.4.3 Blacklisting is working as expected

**Status:** tested and verified 

6.4.4 The smart contract is coded according to the newest standards and in a secure way.

**Status:** tested and verified 

## 6.5 Unit Tests

### Blacklisting

- ✓ Only blacklister is allowed to blacklist. (61ms)
- ✓ Blacklister can add to blacklist.

### Burn of tokens

- ✓ Tokens are burned.
- ✓ Tokens are rounded for redemption of less than a cent.

### Token decimal

- ✓ Decimal should be 6

### Minting

- ✓ Only minter allowed to mint. (49ms)
- ✓ Minter can mint.
- ✓ Minter cannot mint for blacklisted account.
- ✓ Minter cannot mint for a redemption address account.

### Redemption address count upper bound update mechanism

- ✓ Value is changed in upgraded contract.

### Total supply

- ✓ Total supply starts with 0.
- ✓ Total increases accordingly.
- ✓ Total decreases after burn.

### Transfers of tokens

- ✓ Tokens are transferred from PSU1 to PSU2.
- ✓ Tokens transfer from blacklisted PSU1 to PSU2 fails
- ✓ Tokens transfer from PSU1 to blacklisted PSU2 fails
- ✓ Tokens approval from blacklisted owner fails.
- ✓ Tokens approval for blacklisted spender fails.
- ✓ Tokens approval when contract is paused fails.
- ✓ Approved tokens transfer of blacklisted spender fails. (39ms)

- ✓ Approved tokens transfer of blacklisted sender fails. (39ms)
- ✓ Approved tokens transfer for blacklisted recipient fails. (44ms)
- ✓ Tokens are not rounded to penny for normal transfer.

#### Unblacklisting

- ✓ Only unblacklister is allowed to unblacklist.
- ✓ Unblacklister can remove from blacklist.

25 passing (3s)

### 6.5.1 Unit Tests Coverage

File	% Stmts	% Branch	% Funcs	% Lines
Ethereum/	88.37	95	82.35	88.37
Blacklistable.sol	100	100	100	100
TIStablecoinV010.sol	86.49	95	76.92	86.49
All files	88.37	95	82.35	88.37

## 7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, no critical, no high, one medium, two low and two informational issues have been found, after the manual and automated security testing.

Update (05.10.2022): The TrueLayer team fixed all issues and passed all tests within the re-check.

## 8. Mainnet Deployment

PENDING

## 9. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive web3 solutions. Their services include web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secured the smart contracts of 1Inch, POA Network, Unicrypt, LUKSO among numerous other top DeFi projects.

Chainsulting currently secures [\\$100 billion](#) in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: <https://chainsulting.de>

### How We Work



**1** -----

#### PREPARATION

Supply our team with audit ready code and additional materials



**2** -----

#### COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



**3** -----

#### AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



**4** -----

#### FIXES

Your development team applies fixes while consulting with our auditors on their safety.



**5** -----

#### REPORT

We check the applied fixes and deliver a full report on all steps done.