Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	TrueUSD (TUSD) Token
Website:	tusd.io
Platform:	Ethereum
Language:	Solidity
Date:	March 2nd, 2024

Table of contents

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the TrueUSD token smart contract from tusd.io was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on March 2nd, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- This Solidity code defines the TrueUSD (TUSD) token contract, an ERC20-compliant token with additional functionalities such as burning, blacklisting, and proof of reserve. Let's break down the main components and functionalities:
    - **Interfaces:**
        - **IERC20:** Defines the standard ERC20 token interface.
        - **ITrueCurrency:** Interface for TrueCurrency token functionalities like minting, burning, blacklisting, etc.
        - **AggregatorV3Interface:** Interface for Chain Link Price Feeds used for proof of reserve.
    - **Libraries:**
        - **SafeMath:** Library for safe mathematical operations to prevent overflows and underflows.
        - **Address:** Library to check if an address is a contract and handle low-level calls.
    - **ProxyStorage:** Contract to store common state variables and mappings used by proxy contracts.
    - **ClaimableOwnable:** Contract providing functionality for ownership transfer.
    - **ProxyStorage:** Contract to store common state variables and mappings used by proxy contracts.
    - **ERC20:** Abstract contract implementing the ERC20 standard token functionality.

- **ReclaimerToken:** Abstract contract extending ERC20 with functionality to reclaim ETH and ERC20 tokens stuck in the contract.
- **BurnableTokenWithBounds:** Abstract contract extending ReclaimerToken with functionality for burning tokens within specified bounds.
- **TrueCurrency:** Abstract contract extending BurnableTokenWithBounds with additional functionalities like blacklisting, minting, and transferring with specific checks.
- **TrueCurrencyWithProofOfReserve:** Abstract contract extending TrueCurrency with proof of reserve functionality. It ensures that token minting is backed by sufficient reserves based on a Chain Link price feed.
- **TrueUSD:** The main TrueUSD token contract, implementing the TrueCurrencyWithProofOfReserve contract and specifying the token name, symbol, and decimals.

- The TrueUSD token contract includes features such as blacklisting addresses, burning tokens within specified bounds, and ensuring that token minting is backed by sufficient reserves based on a Chain Link price feed.
- TrueUSD is the top-level ERC20 contract, with features like blacklist and redemption addresses, and a Proof-of-Reserves feed check.
- It is owned by the token controller, responsible for minting and admin.
- The platform tracks coin burning and returns the equivalent amount of money.

Audit scope

Name	**Code Review and Security Analysis Report for TrueUSD (TUSD) Token Smart Contract**
Platform	**Ethereum**
File	TrueUSD.sol
Smart Contract Code	0xb650eb28d35691dd1bd481325d40e65273844f9b
Audit Date	March 2nd, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: TrueUSD • Symbol: TUSD • Decimals: 18 • Rounding: 2	**YES, This is valid.**
Ownership control: • Sets a new feed address. • Sets the feed's heartbeat expectation. • Enable / Disable the Proof of Reserve check. • Set blacklisted status for the account. • Set canBurn status for the account. • Change the minimum and maximum amount that can be burned at once _to address to send ether balance to. • Send all ether balance in the contract to another address. • Send all token balances of an arbitrary erc20 token in the contract to another address. • The current owner can transfer ownership of the contract to a new account.	**YES, This is valid. We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contract is **“Secured”**. Also, this contract contains owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. A general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 1 low, and 4 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Not Detected
Fee Check	No
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	Not Detected
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Yes
Blacklist Check	Yes
Can Mint?	Yes
Is it a Proxy?	Yes
Can Take Ownership?	Not Detected
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract file. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in TUSD Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the TUSD Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a TUSD Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that are based on well known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

TrueUSD.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	decimals	write	Passed	No Issue
3	rounding	write	Passed	No Issue
4	name	write	Passed	No Issue
5	symbol	write	Passed	No Issue
6	_mint	internal	Timestamp issue	Refer Audit Findings
7	setChainReserveFeed	external	access only Owner	No Issue
8	setChainReserveHeartbeat	external	access only Owner	No Issue
9	disableProofOfReserve	external	access only Owner	No Issue
10	enableProofOfReserve	external	access only Owner	No Issue
11	mint	external	access only Owner	No Issue
12	setBlacklisted	external	access only Owner	No Issue
13	setCanBurn	external	access only Owner	No Issue
14	_transfer	internal	Passed	No Issue
15	_approve	internal	Passed	No Issue
16	_burn	internal	Passed	No Issue
17	isRedemptionAddress	internal	Passed	No Issue
18	burn	external	Passed	No Issue
19	setBurnBounds	external	access only Owner	No Issue
20	_burn	internal	Passed	No Issue
21	reclaimEther	external	Passed	No Issue
22	reclaimToken	external	Passed	No Issue
23	name	write	Passed	No Issue
24	symbol	write	Passed	No Issue
25	decimals	write	Passed	No Issue
26	totalSupply	read	Passed	No Issue
27	balanceOf	read	Passed	No Issue
28	transfer	write	Passed	No Issue
29	allowance	read	Passed	No Issue
30	approve	write	Passed	No Issue
31	transferFrom	write	Passed	No Issue
32	increaseAllowance	write	Passed	No Issue
33	decreaseAllowance	write	Passed	No Issue
34	_transfer	internal	Passed	No Issue
35	_mint	internal	Passed	No Issue
36	_burn	internal	Passed	No Issue
37	_approve	internal	Passed	No Issue
38	_beforeTokenTransfer	internal	Passed	No Issue
39	onlyOwner	modifier	Passed	No Issue
40	onlyPendingOwner	modifier	Passed	No Issue

41	transferOwnership	external	Critical operation lacks event log, Missing zero check	Refer Audit Findings
42	claimOwnership	external	access only Pending Owner	No Issue
43	_msgSender	internal	Passed	No Issue
44	_msgData	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No high-severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Critical operation lacks event log: **ClaimableOwnable.sol**

```
function transferOwnership(address newOwner) external onlyOwner {
        pendingOwner = newOwner;
    }
```

Detecting missing events for critical access control parameters transferOwnership() has no event, so it is difficult to track off-chain owner changes.

Resolution: We suggest emitting an event for critical parameter changes.

Very Low / Informational / Best practices:

(1) Timestamp issue: **TrueCurrencyWithProofOfReserve.sol**

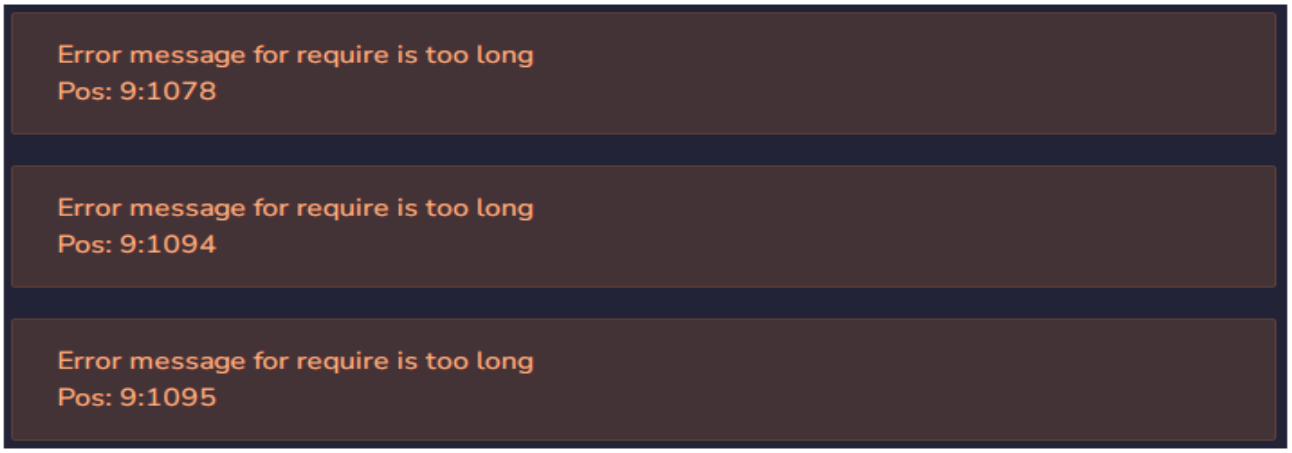
```
TrueCurrencyWithProofOfReserve._mint(address,uint256) (TrueUSD.sol#1271-1296) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= updatedAt,TrueCurrency: invalid PoR updatedAt) (TrueUSD.sol#12
86)
        - require(bool,string)(block.timestamp.sub(updatedAt) <= chainReserveHeartbeat,TrueCurrency: PoR answer
too old) (TrueUSD.sol#1289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Resolution: We suggest avoiding relying on block.timestamp.

(2) Error message for require is too long:

BurnableTokenWithBounds.sol



Ethereum has a gas limit for each block. This limit includes the gas used by all transactions and contract executions within that block. When a required statement fails, it results in an exception, and the error message, along with the gas used up to that point, is included in the transaction's revert message.

Resolution: We suggest writing short and clear messages in the required statements.

(3) Use the latest solidity version: **TrueUSD.sol**

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use versions greater than 0.8.14.

(4) Missing zero check: **ClaimableOwnable.sol**

```
function transferOwnership(address newOwner) external onlyOwner {
    pendingOwner = newOwner;
}
```

Detects missing zero address validation.

Resolution: We suggest checking that the address is not zero.

Centralization

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key is compromised, then it would create trouble. Following are Admin functions:

TrueCurrencyWithProofOfReserve.sol

- setChainReserveFeed: Sets a new feed address by the owner.
- setChainReserveHeartbeat: Sets the feed's heartbeat expectation by the owner.
- disableProofOfReserve: Disable Proof of Reserve check by the owner.
- enableProofOfReserve: Enable Proof of Reserve check by the owner.

TrueCurrency.sol

- mint: Mint tokens by the owner.
- setBlacklisted: Set blacklisted status for the account by the owner.
- setCanBurn: Set the canBurn status for the account by the owner.

BurnableTokenWithBounds.sol

- setBurnBounds: Change the minimum and maximum amount that can be burned at once _to address to send ether balance to by the owner.

ReclaimerToken.sol

- reclaimEther: Send all ether balance in the contract to another address by the owner.
- reclaimToken: Send all token balance of an arbitrary erc20 token in the contract to another address by the owner.

ClaimableOwnable.sol

- transferOwnership: Current owner can transfer ownership of the contract to a new account.
- claimOwnership: Allows the pending owner address to finalize the transfer by the pending owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 low and 4 informational issues in the smart contract. And those issues are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

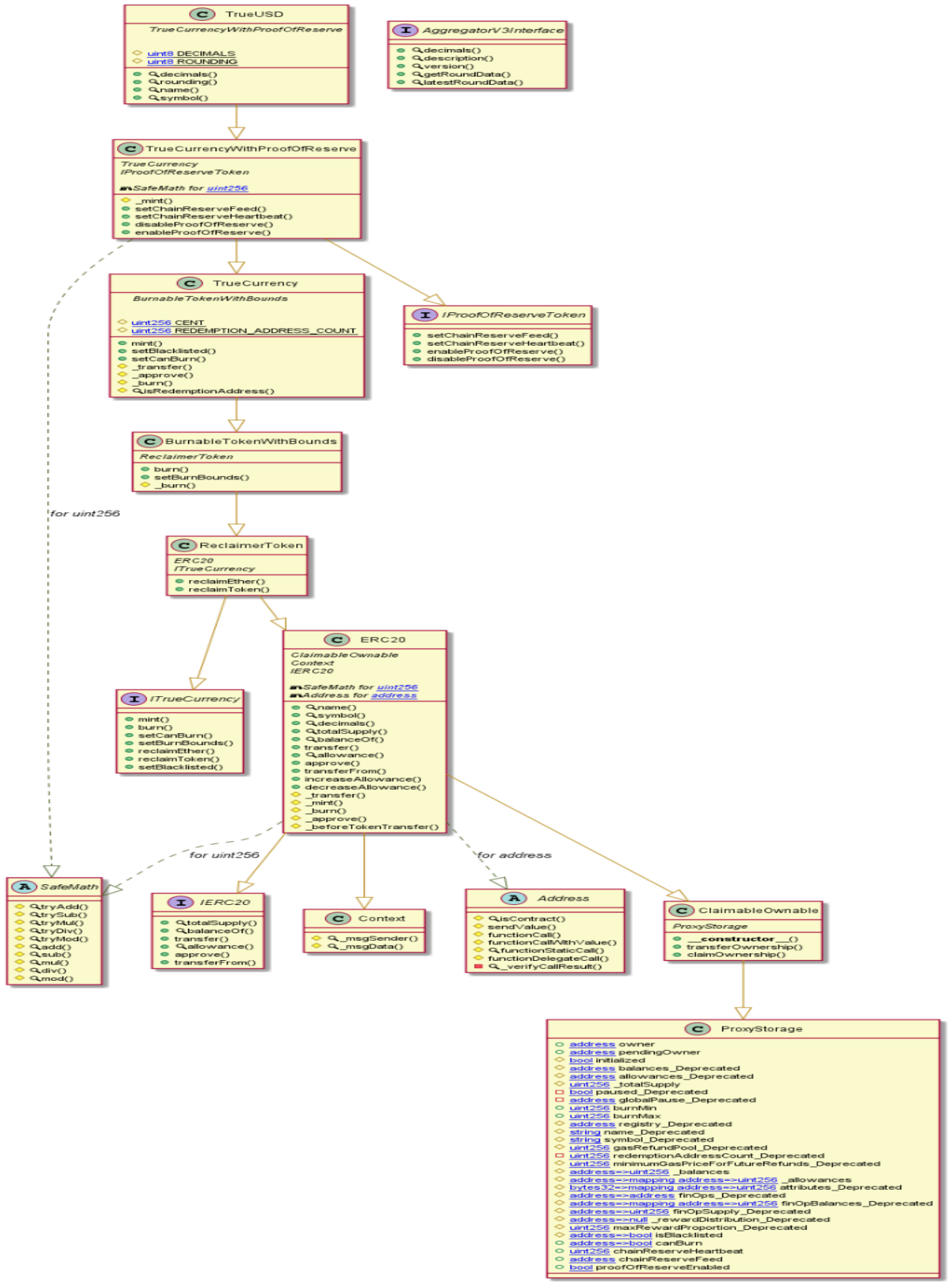
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - TrueUSD (TUSD) Token

TrueUSD Diagram



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> TrueUSD.sol

```
ReclaimerToken.reclaimToken(IERC20,address) (TrueUSD.sol#1016-1019) ignores return value by token.transfer(_to,balance) (TrueUSD.sol#1018)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

ERC20.allowance(address,address).owner (TrueUSD.sol#813) shadows:
        - ProxyStorage.owner (TrueUSD.sol#566) (state variable)
ERC20._approve(address,address,uint256).owner (TrueUSD.sol#961) shadows:
        - ProxyStorage.owner (TrueUSD.sol#566) (state variable)
TrueCurrency._approve(address,address,uint256).owner (TrueUSD.sol#1241) shadows:
        - ProxyStorage.owner (TrueUSD.sol#566) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

ClaimableOwnable.transferOwnership(address) (TrueUSD.sol#686-688) should emit an event for:
        - pendingOwner = newOwner (TrueUSD.sol#687)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

TrueCurrencyWithProofOfReserve.setChainReserveFeed(address).newFeed (TrueUSD.sol#1413) lacks a zero-check on :
                - chainReserveFeed = newFeed (TrueUSD.sol#1415)
ReclaimerToken.reclaimEther(address)._to (TrueUSD.sol#1006) lacks a zero-check on :
                - _to.transfer(address(this).balance) (TrueUSD.sol#1007)
ClaimableOwnable.transferOwnership(address).newOwner (TrueUSD.sol#686) lacks a zero-check on :
                - pendingOwner = newOwner (TrueUSD.sol#687)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

TrueCurrencyWithProofOfReserve._mint(address,uint256) (TrueUSD.sol#1381-1406) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp >= updatedAt,TrueCurrency: invalid PoR updatedAt) (TrueUSD.sol#1396)
        - require(bool,string)(block.timestamp.sub(updatedAt) <= chainReserveHeartbeat,TrueCurrency: PoR answer too old) (TrueUSD.sol#1399)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Address.isContract(address) (TrueUSD.sol#387-396) uses assembly
        - INLINE ASM (TrueUSD.sol#394)
Address._verifyCallResult(bool,bytes,string) (TrueUSD.sol#532-549) uses assembly
        - INLINE ASM (TrueUSD.sol#541-544)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._verifyCallResult(bool,bytes,string) (TrueUSD.sol#532-549) is never used and should be removed
Address.functionCall(address,bytes) (TrueUSD.sol#440-442) is never used and should be removed
Address.functionCall(address,bytes,string) (TrueUSD.sol#450-452) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (TrueUSD.sol#465-467) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (TrueUSD.sol#475-482) is never used and should be removed
Address.functionDelegateCall(address,bytes) (TrueUSD.sol#514-516) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (TrueUSD.sol#524-530) is never used and should be removed
Address.functionStaticCall(address,bytes) (TrueUSD.sol#490-492) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (TrueUSD.sol#500-506) is never used and should be removed
Address.isContract(address) (TrueUSD.sol#387-396) is never used and should be removed
Address.sendValue(address,uint256) (TrueUSD.sol#414-420) is never used and should be removed
Context._msgData() (TrueUSD.sol#129-132) is never used and should be removed
SafeMath.div(uint256,uint256) (TrueUSD.sol#279-282) is never used and should be removed
SafeMath.div(uint256,uint256,string) (TrueUSD.sol#334-337) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (TrueUSD.sol#354-357) is never used and should be removed
SafeMath.mul(uint256,uint256) (TrueUSD.sol#260-265) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (TrueUSD.sol#168-172) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (TrueUSD.sol#204-207) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (TrueUSD.sol#214-217) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (TrueUSD.sol#189-197) is never used and should be removed
SafeMath.trySub(uint256,uint256) (TrueUSD.sol#179-182) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version0.6.10 (TrueUSD.sol#9) allows old versions
solc-0.6.10 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Low level call in Address.sendValue(address,uint256) (TrueUSD.sol#414-420):
        - (success) = recipient.call{value: amount}() (TrueUSD.sol#418)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (TrueUSD.sol#475-482):
        - (success,returndata) = target.call{value: value}(data) (TrueUSD.sol#480)
Low level call in Address.functionStaticCall(address,bytes,string) (TrueUSD.sol#500-506):
        - (success,returndata) = target.staticcall(data) (TrueUSD.sol#504)
Low level call in Address.functionDelegateCall(address,bytes,string) (TrueUSD.sol#524-530):
        - (success,returndata) = target.delegatecall(data) (TrueUSD.sol#528)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable ProxyStorage.balances_Deprecated (TrueUSD.sol#571) is not in mixedCase
Variable ProxyStorage.allowances_Deprecated (TrueUSD.sol#572) is not in mixedCase
Variable ProxyStorage._totalSupply (TrueUSD.sol#574) is not in mixedCase
Variable ProxyStorage.paused_Deprecated (TrueUSD.sol#576) is not in mixedCase
Variable ProxyStorage.globalPause_Deprecated (TrueUSD.sol#577) is not in mixedCase
Variable ProxyStorage.registry_Deprecated (TrueUSD.sol#582) is not in mixedCase
Variable ProxyStorage.name_Deprecated (TrueUSD.sol#584) is not in mixedCase
Variable ProxyStorage.symbol_Deprecated (TrueUSD.sol#585) is not in mixedCase
Variable ProxyStorage.gasRefundPool_Deprecated (TrueUSD.sol#587) is not in mixedCase
Variable ProxyStorage.redemptionAddressCount_Deprecated (TrueUSD.sol#588) is not in mixedCase
Variable ProxyStorage.minimumGasPriceForFutureRefunds_Deprecated (TrueUSD.sol#589) is not in mixedCase
Variable ProxyStorage._balances (TrueUSD.sol#591) is not in mixedCase
Variable ProxyStorage._allowances (TrueUSD.sol#592) is not in mixedCase
Variable ProxyStorage.attributes_Deprecated (TrueUSD.sol#593) is not in mixedCase
Variable ProxyStorage.finOps_Deprecated (TrueUSD.sol#596) is not in mixedCase
Variable ProxyStorage.finOpBalances_Deprecated (TrueUSD.sol#597) is not in mixedCase
Variable ProxyStorage.finOpSupply_Deprecated (TrueUSD.sol#598) is not in mixedCase
Variable ProxyStorage._rewardDistribution_Deprecated (TrueUSD.sol#606) is not in mixedCase
Variable ProxyStorage.maxRewardProportion_Deprecated (TrueUSD.sol#607) is not in mixedCase
Parameter ReclaimerToken.reclaimEther(address)._to (TrueUSD.sol#1006) is not in mixedCase
Parameter ReclaimerToken.reclaimToken(IERC20,address)._to (TrueUSD.sol#1016) is not in mixedCase
Parameter BurnableTokenWithBounds.setBurnBounds(uint256,uint256)._min (TrueUSD.sol#1078) is not in mixedCase
Parameter BurnableTokenWithBounds.setBurnBounds(uint256,uint256)._max (TrueUSD.sol#1078) is not in mixedCase
Parameter TrueCurrency.setBlacklisted(address,bool)._isBlacklisted (TrueUSD.sol#1191) is not in mixedCase
Parameter TrueCurrency.setCanBurn(address,bool)._canBurn (TrueUSD.sol#1206) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Redundant expression "this (TrueUSD.sol#130)" inContext (TrueUSD.sol#124-133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

TrueUSD.slitherConstructorConstantVariables() (TrueUSD.sol#1465-1485) uses literals with too many digits:
        - REDEMPTION_ADDRESS_COUNT = 0x100000 (TrueUSD.sol#1147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

ProxyStorage.initialized (TrueUSD.sol#569) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.balances_Deprecated (TrueUSD.sol#571) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.allowances_Deprecated (TrueUSD.sol#572) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.registry_Deprecated (TrueUSD.sol#582) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.name_Deprecated (TrueUSD.sol#584) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.symbol_Deprecated (TrueUSD.sol#585) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.gasRefundPool_Deprecated (TrueUSD.sol#587) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.minimumGasPriceForFutureRefunds_Deprecated (TrueUSD.sol#589) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.attributes_Deprecated (TrueUSD.sol#593) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.finOps_Deprecated (TrueUSD.sol#596) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.finOpBalances_Deprecated (TrueUSD.sol#597) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.finOpSupply_Deprecated (TrueUSD.sol#598) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage._rewardDistribution_Deprecated (TrueUSD.sol#606) is never used in TrueUSD (TrueUSD.sol#1465-1485)
ProxyStorage.maxRewardProportion_Deprecated (TrueUSD.sol#607) is never used in TrueUSD (TrueUSD.sol#1465-1485)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

ProxyStorage.allowances_Deprecated (TrueUSD.sol#572) should be constant
ProxyStorage.balances_Deprecated (TrueUSD.sol#571) should be constant
ProxyStorage.initialized (TrueUSD.sol#569) should be constant
ProxyStorage.maxRewardProportion_Deprecated (TrueUSD.sol#607) should be constant
ProxyStorage.minimumGasPriceForFutureRefunds_Deprecated (TrueUSD.sol#589) should be constant
ProxyStorage.name_Deprecated (TrueUSD.sol#584) should be constant
ProxyStorage.registry_Deprecated (TrueUSD.sol#582) should be constant
ProxyStorage.symbol_Deprecated (TrueUSD.sol#585) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
TrueUSD.sol analyzed (15 contracts with 84 detectors), 87 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

TrueUSD.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 475:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

more

Pos: 541:16:

Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.

more

Pos: 528:50:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function TrueUSD.transfer is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 805:4:

Gas costs:

Gas requirement of function TrueUSD.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1064:4:

Constant/View/Pure functions:

TrueCurrency._burn(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1256:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1446:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 336:15:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

TrueUSD.sol

```
Error message for require is too long
Pos: 9:262
Error message for require is too long
Pos: 9:418
Error message for require is too long
Pos: 9:475
Error message for require is too long
Pos: 9:500
Error message for require is too long
Pos: 9:524
Explicitly mark visibility of state
Pos: 5:568
Explicitly mark visibility of state
Pos: 5:570
Explicitly mark visibility of state
Pos: 5:571
Explicitly mark visibility of state
Pos: 5:573
Explicitly mark visibility of state
Pos: 5:581
Explicitly mark visibility of state
Pos: 5:583
Explicitly mark visibility of state
Pos: 5:584
Explicitly mark visibility of state
Pos: 5:586
Explicitly mark visibility of state
Pos: 5:588
Explicitly mark visibility of state
Pos: 5:590
Explicitly mark visibility of state
Pos: 5:591
Explicitly mark visibility of state
Pos: 5:592
Explicitly mark visibility of state
Pos: 5:595
Explicitly mark visibility of state
Pos: 5:596
Explicitly mark visibility of state
Pos: 5:597
Explicitly mark visibility of state
Pos: 5:605
Explicitly mark visibility of state
```

```
Pos: 5:606
Explicitly mark visibility of state
Pos: 5:608
Error message for require is too long
Pos: 9:897
Error message for require is too long
Pos: 9:898
Error message for require is too long
Pos: 9:938
Error message for require is too long
Pos: 9:961
Error message for require is too long
Pos: 9:1095
Explicitly mark visibility of state
Pos: 5:1145
Explicitly mark visibility of state
Pos: 5:1146
Error message for require is too long
Pos: 9:1387
Error message for require is too long
Pos: 9:1391
Error message for require is too long
Pos: 9:1395
Avoid making time-based decisions in business logic
Pos: 17:1395
Avoid making time-based decisions in business logic
Pos: 17:1398
Error message for require is too long
Pos: 9:1403
Error message for require is too long
Pos: 9:1445
Error message for require is too long
Pos: 9:1446
Compiler version 0.6.10 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1455
Explicitly mark visibility of state
Pos: 5:1465
Explicitly mark visibility of state
Pos: 5:1466
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io