



TrustToken

TrueFi Protocol

SMART CONTRACT AUDIT

21.12.2021

Made in Germany by Chainsulting.de



Table of contents

1. Disclaimer.....	3
2. About the Project and Company	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology	7
4.2 Used Code from other Frameworks/Smart Contracts	8
4.3 Tested Contract Files	9
4.4 Metrics / CallGraph.....	10
4.5 Metrics / Source Lines & Risk.....	11
4.6 Metrics / Capabilities	12
4.7 Metrics / Source Unites in Scope	13
5. Scope of Work.....	15
5.1 Manual and Automated Vulnerability Test.....	16
5.2 Verify claims	18
6. Executive Summary.....	26

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 TrustToken Inc. 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 (05.12.2021)	Layout
0.2 (10.12.2021)	Test Deployment
0.5 (11.12.2021)	Automated Security Testing Manual Security Testing
0.6 (13.12.2021)	Testing SWC Checks
0.7 (15.12.2021)	Verify Claims
0.9 (20.12.2021)	Summary and Recommendation
1.0 (22.12.2021)	Final document

2. About the Project and Company



Company address:

TrustToken Inc.
234 S Main Street Suite 7 Willits
California 95490
United States of America

Website: <https://www.trusttoken.com>

Twitter: <https://twitter.com/TrustToken>

Reddit: <https://www.reddit.com/r/TrustToken>

Telegram: <https://t.me/jointruefi>

Discord: <https://bit.ly/chattruefi>

LinkedIn: <https://www.linkedin.com/company/trusttoken>

Facebook: <https://www.facebook.com/TrustToken/>

Medium: <https://trusttokenteam.medium.com>

YouTube: <https://www.youtube.com/channel/UCePpU7NPWENI6rdmFb7HALA>

2.1 Project Overview

TrustToken is a platform to create asset-backed tokens that you can easily buy and sell around the world. For example, gold to gold tokens or dollar to dollar tokens. The company's first asset token is TrueUSD, a stablecoin that you can redeem 1-for-1 for US dollars. TrustToken was founded in 2017 and consists of a team from Stanford, UC Berkeley, Airbnb, Goldman Sachs, PayPal, and Google, and is backed by a16z crypto, BlockTower Capital, Danhua Capital, Founders Fund Angel, GGV Capital, Jump Capital, Stanford-StartX, and others.

TrustToken has launched TrueFi, the protocol for uncollateralized lending, powered by the first ever on-chain credit scores and governed by holders of the TRU token. At launch on November 21st, 2020, TrueFi provided for (a) vetted borrowers to request loans denominated in TrueUSD ("TUSD"), (b) TRU Stakers to assess the creditworthiness of loans, (c) and TrueUSD lenders to earn attractive APY & TRU incentives on stablecoins loaned on the protocol.

Since that launch, TrueFi has evolved rapidly following a public roadmap, undergone two major protocol upgrades, started decentralizing protocol governance via Snapshot, and exceeded \$200 million in loan originations with zero defaults — making TrueFi DeFi's first and leading uncollateralized lending protocol. This litepaper was updated July 2021 to include these milestones & reflect changes in the design of the protocol. While much of DeFi's success has been built on overcollateralized lending, uncollateralized lending and bringing true credit scoring to crypto is widely seen as the next transformative step for DeFi.

The traditional unsecured lending market makes up a \$11 trillion global industry — yet none of that lending had come on-chain until TrueFi completed DeFi's first uncollateralized loan in 2020. Because uncollateralized lending provides an opportunity for lenders to earn higher long-term returns than secured lending, and for borrowers to maximize their capital efficiency, we believe on-chain, collateral-free lending will ultimately far outpace DeFi's existing collateralized lending market.

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.

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

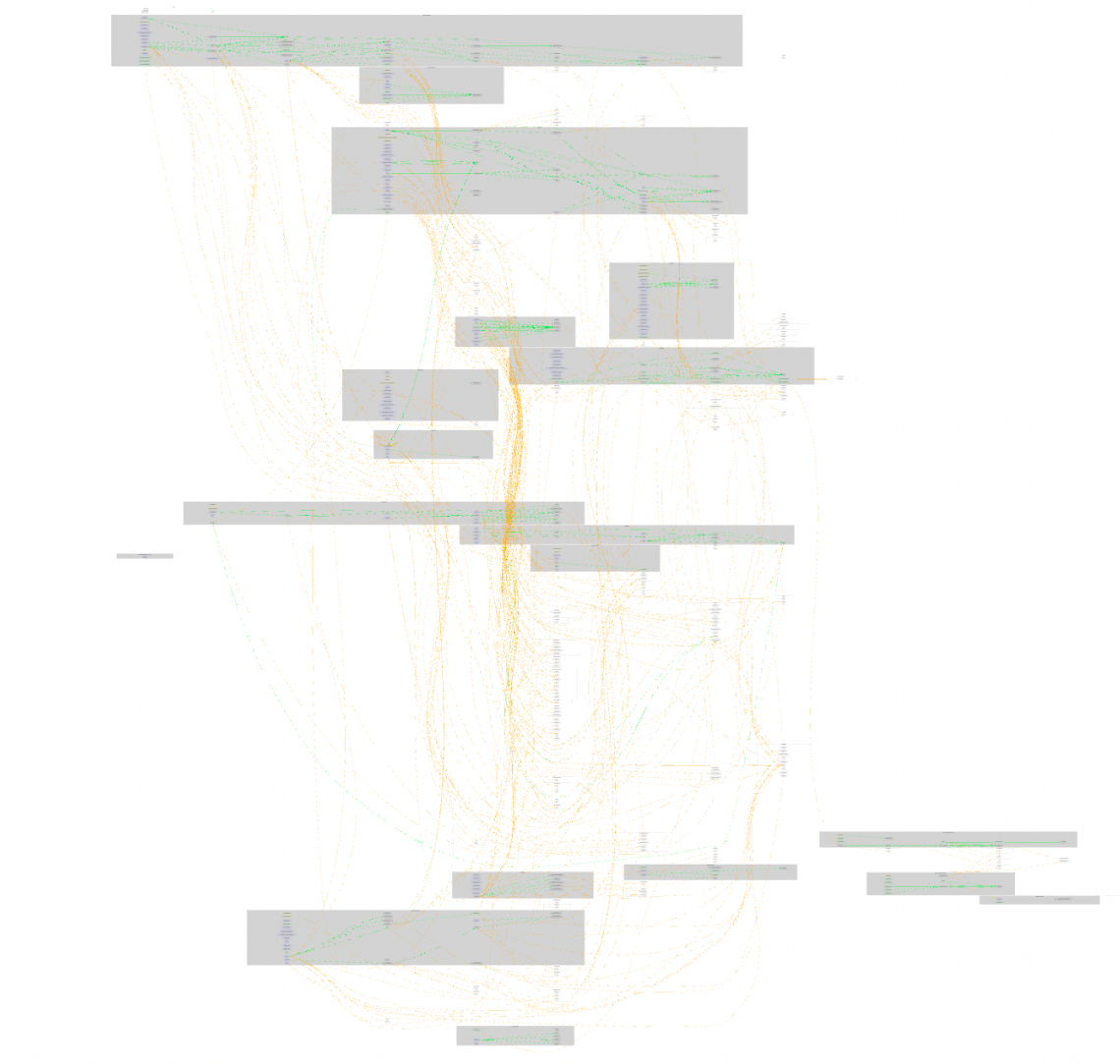
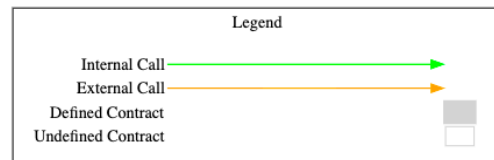
Dependency / Import Path	Source
@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol	https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol
@openzeppelin/contracts/math/Math.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.0/contracts/math/Math.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.0/contracts/math/SafeMath.sol
@openzeppelin/contracts/proxy/Clones.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.0/contracts/proxy/Clones.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.0/contracts/token/ERC20/SafeERC20.sol

4.3 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/truefi2/LineOfCreditAgency.sol	834449374e638264eae67ce9b948325a
./contracts/truefi2/DeficiencyToken.sol	ac148aa7f66bbfa6e6bcd6da49904a87
./contracts/truefi2/StakingVault.sol	4e6a47c2b8a563855cff419599f2a206
./contracts/truefi2/Liquidator2.sol	bb62ab3c77016e7320c5fed24ed60321
./contracts/truefi2/TrueFiPool2.sol	40e52b22d3db04d776e8ffb3c83599c2
./contracts/truefi2/TrueFiCreditOracle.sol	95aaded6dae45c6aba37c10178c9628f
./contracts/truefi2/FixedTermLoanAgency.sol	52e24e2965f4a2c0be1ec696306a0689
./contracts/truefi2/LoanFactory2.sol	6aa956794772cbbdcdf77915f746e618
./contracts/truefi2/TimeAveragedBaseRateOracle.sol	0759fa38e5cd4537848b4b1c388f4d29
./contracts/truefi2/SAFU.sol	f2289ef54f5c411a6a04181a4190c151
./contracts/truefi2/PoolFactory.sol	8059d4ee30a3343209a850c4a0c2b6b5
./contracts/truefi2/RateModel.sol	137383384f089ef89e5257e528b70e9b
./contracts/truefi2/SpotBaseRateOracle.sol	1b39484dfb7072f4d2c7d99bedbf3a7b
./contracts/truefi2/FixedTermLoan.sol	bd7e1fc0ba9251c4e0dfff464839a562
./contracts/truefi2/BorrowingMutex.sol	c64dd443758ac2820789542519f12662
./contracts/truefi2/DebtToken.sol	de28b5d9d6545d484977d67cdab207c8
./contracts/truefi2/oracles/TimeAveragedTruPriceOracle.sol	5a96445c2ece14e32da22488e287c19c
./contracts/truefi2/oracles/ChainlinkTruOracle.sol	8c542b7ad1d69932644d400b4b1fe57f

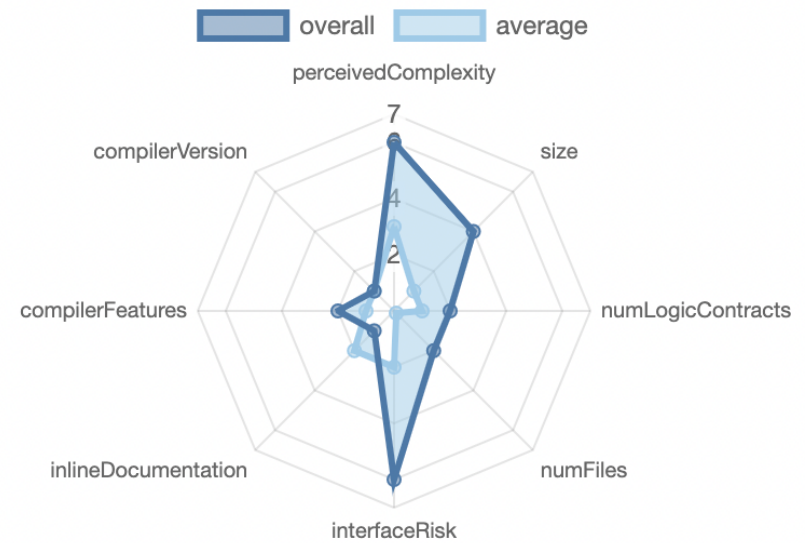
4.4 Metrics / CallGraph













View full version: <https://chainsulting.de/wp-content/uploads/2021/12/solidity-metrics-truefi.html>

4.5 Metrics / Source Lines & Risk

source comment single block mixed
empty todo blockEmpty





4.6 Metrics / Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
0.6.10		ABIEncoderV2			
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRecover	 New/Create/Create2
					yes → NewContract:DeficiencyToken → NewContract:OwnedProxyWithReferenc e

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				
227	0				
External	Internal	Private	Pure	View	
140	193	8	15	91	

StateVariables

Total	 Public
173	147

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSL OC	Comment Lines	Complex. Score	Capabilities
	contracts/truefi2/LineOfCreditAgency.sol	1	1	839	780	442	236	351	
	contracts/truefi2/DeficiencyToken.sol	1		44	44	27	9	22	
	contracts/truefi2/StakingVault.sol	1		102	97	69	7	52	
	contracts/truefi2/Liquidator2.sol	1		242	234	135	61	120	
	contracts/truefi2/TrueFiPool2.sol	1		755	741	353	277	370	
	contracts/truefi2/TrueFiCreditOracle.sol	1		167	167	75	63	65	
	contracts/truefi2/FixedTermLoanAgency.sol	1	1	532	496	268	157	243	
	contracts/truefi2/LoanFactory2.sol	1		209	193	119	33	96	
	contracts/truefi2/TimeAveragedBaseRateOracle.sol	1		178	166	76	64	56	
	contracts/truefi2/SAFU.sol	1		237	233	145	53	190	 
	contracts/truefi2/PoolFactory.sol	1		362	348	170	121	188	
	contracts/truefi2/RateModel.sol	1	1	392	344	189	113	173	

Typ e	File	Logic Contra cts	Interfaces	Lin es	nLin es	nSL OC	Comm ent Lines	Compl ex. Score	Capabiliti es
	contracts/truefi2/SpotBaseRateOracle.sol	1	_____	39	39	17	16	10	_____
	contracts/truefi2/FixedTermLoan.sol	1	_____	274	263	129	93	121	_____
	contracts/truefi2/BorrowingMutex.sol	1	_____	75	75	48	7	41	_____
	contracts/truefi2/DebtToken.sol	1	_____	157	151	73	51	67	_____
	contracts/truefi2/oracles/TimeAveragedTruPriceOracle.sol	1	_____	177	169	81	61	62	_____
	contracts/truefi2/oracles/ChainlinkTruOracle.sol	1	_____	68	60	29	30	33	_____
	Totals	18	3	4849	4600	2445	1452	2260	

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, ...)

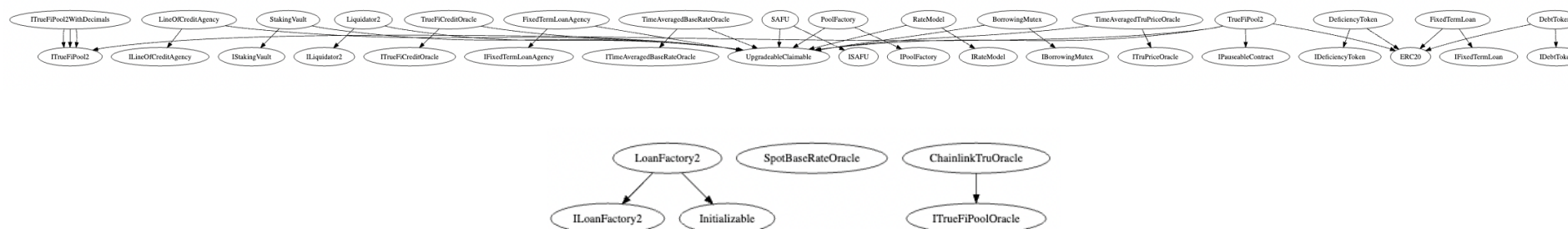
5. Scope of Work

The TrustToken Team provided us with the files that needs to be tested. The scope of the audit are the TrueFi protocol contracts.

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

- The smart contract is coded according to the newest standards and in a secure way
- FixedTermLoanAgency.borrow() is not vulnerable to denial of service.
- The LineOfCreditAgency.buckets mapping stays internally consistent no matter what functions are called.
- It is intractable to manipulate RateModel.rate() down to favor borrowers or up to favor lenders.
- StakingVault.unstake() cannot bring a borrower's total borrowed amount over their borrow limit.
- Once a borrower has defaulted anywhere in the protocol, they can never again withdraw funds from any part of the protocol.
- TrueFi Protocol cannot be effected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzx

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.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES

5.1.1 Public functions should be declared as external

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: BorrowingMutex.sol, DebtToken.sol, FixedTermLoan.sol, LineOfCreditAgency.sol, Liquidator2.sol, PoolFactory.sol, RateModel.sol, SAFU.sol, TimeAveragedBaseOracle.sol, TrueFiCreditOracle.sol, TrueFiPool2.sol, TrueMultiFarm.sol

Attack / Description	Code Snippet	Result/Recommendation
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.	BorrowingMutex.isBanned DebtToken.decimals FixedTermLoan.decimals FixedTermLoanAgency.initialize FixedTermLoanAgency.setBorrowingMutex FixedTermLoanAgency.loans LineOfCreditAgency.initialize LineOfCreditAgency.utilizationAdjustmentRate	We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.


Because memory allocation is expensive, the gas consumption of public functions is higher.	LineOfCreditAgency.borrowLimitAdjustment Liquidator2.initilize PoolFactory.supportedPoolsTVL RateModel.initialize RateModel.fixedTermLoanAdjustment RateModel.effectiveScore RateModel.borrowLimit RateModel.isOverLimit SAFU.initialize SAFU.legacyRedeem SAFU.redeem TimeAveragedBaseRateOracle.setSpotOracle TimeAveragedBaseRateOracle.getTotalsBuffer TimeAveragedBaseRateOracle.getWeeklyAPY TimeAveragedBaseRateOracle.getMonthlyAPY TimeAveragedBaseRateOracle.getYearlyAPY TrueFiCreditOracle.initialize TrueFiCreditOracle.setScore TrueFiCreditOracle.setMaxBorrowLimit TrueFiCreditOracle.setManager TrueFiPool2.decimals TrueMultiFarm.initialize	
--	---	--

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **no Informational issues** in the code of the smart contract.

5.2 Verify claims

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

Status: tested and verified 

5.2.2 FixedTermLoanAgency.borrow() is not vulnerable to denial of service.

Status: tested and verified 

Description: The service of borrowing will only be denied if the require checks are not met. There is no additional case, where the service will be potentially denied in an unintended way.


One-click borrowing

The FixedTermLoanAgency.borrow() combines the functionality of LoanFactory2.createLoanToken(), TrueLender.fund(loan) and LoanToken.withdraw() in a secure way (FixedTermLoanAgency line 327 - 363). Borrowers can take a loan by calling this function and may not take any other loans while they have a fixed term loan outstanding. This is done by locking borrowers address while they have outstanding loans (FixedTermLoanAgency line 357).

```
356 |         IFixedTermLoan loanToken = loanFactory.createLoanToken(pool, borrower, amount, term, apy);
357 |         borrowingMutex.lock(borrower, address(loanToken));
358 |         poolLoans[pool].push(loanToken);
359 |         pool.borrow(amount);
360 |         pool.token().safeTransfer(borrower, amount);
361 |
362 |         emit Funded(address(pool), address(loanToken), amount);
```

Cannot verify functionality of TrueRatingAgencyV2.submit(loan). There is no submit function in TrueRatingAgencyV2.

5.2.3 The LineOfCreditAgency.buckets mapping stays internally consistent no matter what functions are called.

Status: tested and verified 

Description: The LineOfCreditAgency.buckets mapping stays internally consistent and only changes internally by executing the _pokeSingleBucket function to update state for a single bucket. The changes are written consistent to storage.

Lines of credit

An allowed borrower may take out lines of credit in all supported pools, unless they have a fixed term loan outstanding. The locking mechanism is totally independent from fixed term loan lock, but borrowers can only have one line of debt position at a time.


```
401 |         if ( totalBorrowed == 0 ) {  
402 |             borrowingMutex.lock(msg.sender, address(this));  
403 |         }  
404 |         require(  
405 |             borrowingMutex.locker(msg.sender) == address(this),  
406 |             "LineOfCreditAgency: Borrower cannot open two simultaneous debt positions"  
407 |         );
```

Every 31 days (since last interest repayment), the borrower must repay all interest accrued on each of their lines of credit, or face technical default. Interest rates and borrow limits are variable. They may change block-by-block, as determined by the rate model, described below.

The buckets combine all borrowers with the same credit score together (max of 256 different scores), to avoid hitting the block gas limit when updating all borrowers' cached interest amounts.

```
84 |     /// @dev credit score buckets for each pool  
85 |     mapping(ITrueFiPool2 => CreditScoreBucket[256]) public buckets;
```

5.2.4 It is intractable to manipulate RateModel.rate() down to favor borrowers or up to favor lenders.

Status: tested and verified 

Description: Rates are calculated as the sum of a secured rate (7-day TWAP of Aave USD borrow rates) and adjustments based on the borrower creditworthiness and the lending pool utilization. There is no way to manipulate the rate to favor of borrowers or lenders.

Rate model

Interest rates and borrow limits for {an ongoing line of credit} or {creation of a fixed term loan} fluctuate block-by-block according to a rate model. Rates are the sum of a secured rate (7-day TWAP of Aave USD borrow rates, to smooth out volatility), an unsecured risk premium that's manually set, and adjustments based on:

```
241 | function combinedRate(uint256 partialRate, uint256 __creditScoreAdjustmentRate) public pure override returns (uint256) {
242 |     return min(partialRate.add(__creditScoreAdjustmentRate), MAX_RATE_CAP);
243 | }
```

a) Borrower creditworthiness, as determined by a manually set credit score and TRU staked (see below), and

```
250 | function creditScoreAdjustmentRate(uint8 score) public view override returns (uint256) {
251 |     if (score == 0) {
252 |         return MAX_RATE_CAP; // Cap rate by 500%
253 |     }
254 |     uint256 coefficient = uint256(creditScoreRateConfig.coefficient);
255 |     uint256 power = uint256(creditScoreRateConfig.power);
256 |     return min(coefficient.mul(uint256(MAX_CREDIT_SCORE)**power).div(uint256(score)**power).sub(coefficient), MAX_RATE_CAP);
257 | }
```

b) Lending pool utilization, to encourage borrowing when utilization is low and encourage lending when utilization is high

```
264 | function utilizationAdjustmentRate(ITrueFiPool2 pool, uint256 afterAmountLent) public view override returns (uint256) {
265 |     uint256 liquidRatio = pool.liquidRatio(afterAmountLent);
266 |     if (liquidRatio == 0) {
267 |         // if utilization is at 100 %
268 |         return MAX_RATE_CAP; // Cap rate by 500%
269 |     }
270 |     uint256 coefficient = uint256(utilizationRateConfig.coefficient);
271 |     uint256 power = uint256(utilizationRateConfig.power);
272 |     return min(coefficient.mul(uint256(BASIS_POINTS)**power).div(liquidRatio**power).sub(coefficient), MAX_RATE_CAP);
273 | }
```

Borrow limits are the minimum of:


- a) A manually set max borrower limit (adjusted downward by credit score and up by TRU staked),
- b) A max ratio of the TVL (adjusted downward by credit score)
- c) A max ratio of the particular pool

```

364 function poolBorrowMax(
365     ITrueFiPool2 pool,
366     uint8 score,
367     uint256 maxBorrowerLimit,
368     uint256 stakedTru
369 ) internal view returns (uint256) {
370     uint256 maxTVLLimit = poolFactory.supportedPoolsTVL().mul(borrowLimitConfig.tvllLimitCoefficient).div(BASIS_POINTS);
371     uint256 adjustment = borrowLimitAdjustment(score);
372     uint256 stakedValueInUsd = conservativeStakedValue(stakedTru);
373     uint256 adjustedBorrowerLimit = maxBorrowerLimit.mul(adjustment).div(BASIS_POINTS).add(stakedValueInUsd);
374     uint256 adjustedTVLLimit = maxTVLLimit.mul(adjustment).div(BASIS_POINTS);
375     uint256 creditLimit = min(adjustedBorrowerLimit, adjustedTVLLimit);
376     uint256 poolValueInUsd = pool.oracle().tokenToUsd(pool.poolValue());
377     return min(poolValueInUsd.mul(borrowLimitConfig.poolValueLimitCoefficient).div(BASIS_POINTS), creditLimit);
378 }

```

5.2.5 StakingVault.unstake() cannot bring a borrower's total borrowed amount over their borrow limit.

Status: tested and verified 

Description: Before unstaking an amount from StakingVault, LineOfCreditAgency.isOverProFormaLimit is used to check if the borrowed amount would be more than the borrow limit after unstaking (StakingVault line 68 & 100 and LineOfCreditAgency line 346 - 358). This safety check ensures, that the borrowed amount can never be higher than borrow limit based on the staked amount.

Borrower TRU staking

A borrower may stake TRU tokens for more favourable interest rates or borrow limits. This is converted to USD using a 7-day TWAP of a Chainlink TRU/USD oracle.

```

150 function calculateAveragePrice(uint16 numberOfValues) public view returns (uint256) {
151     require(numberOfValues > 0, "TimeAveragedTruPriceOracle: Number of values should be greater than 0");
152     require(numberOfValues < bufferSize(), "TimeAveragedTruPriceOracle: Number of values should be less than buffer size");
153
154     uint16 _currIndex = totalsBuffer.currIndex;
155     uint16 startIndex = (_currIndex + bufferSize() - numberOfValues) % bufferSize();
156
157     if (totalsBuffer.timestamps[startIndex] == 0) {
158         require(_currIndex > 0, "TimeAveragedTruPriceOracle: Cannot use buffer before any update call");
159         startIndex = 0;
160     }
161
162     uint256 diff = totalsBuffer.runningTotals[_currIndex].sub(totalsBuffer.runningTotals[startIndex]);
163     uint256 dt = totalsBuffer.timestamps[_currIndex].sub(totalsBuffer.timestamps[startIndex]);
164     return diff.div(dt);
165 }

```

In case of default, slashing the staked TRU is the protocol's first line of defense to recover funds for the lending pools.


```
76 function slash(address borrower) external override returns (uint256) {
77     require(msg.sender == address(liquidator), "StakingVault: Caller is not the liquidator");
78     uint256 slashedAmount = stakedAmount[borrower];
79     if (slashedAmount == 0) {
80         return 0;
81     }
82     require(borrowingMutex.isBanned(borrower), "StakingVault: Borrower has to be banned");
83
84     stakedAmount[borrower] = 0;
85     stakedToken.safeTransfer(msg.sender, slashedAmount);
86     emit Slashed(borrower, slashedAmount);
87     return slashedAmount;
88 }
```

For fixed term loans, the staked amount is locked in the StakingVault until the loan has been repaid. It is secured by locking the unstake function with borrowing mutex (line 94 - 96). For lines of credit, the staked amount may be withdrawn at any time and thereby adjusts the borrow limit and interest rate (line 97 - 99).

```
90 function canUnstake(address borrower, uint256 amount) public view returns (bool) {
91     if (amount > stakedAmount[borrower]) {
92         return false;
93     }
94     if (borrowingMutex.isUnlocked(borrower)) {
95         return true;
96     }
97     if (borrowingMutex.locker(borrower) != address(lineOfCreditAgency)) {
98         return false;
99     }
100     return !lineOfCreditAgency.isOverProFormaLimit(borrower, stakedAmount[borrower].sub(amount));
101 }
```

To fit within LineOfCreditAgency's bucketing design, the effect on rates is implemented as a change to the effective score used as input to the credit score adjustment curve. That is, with 0 TRU staked we take the borrower's score directly as input, and with maximum TRU staked we pass in a perfect effective credit score of 255. The effect of TRU staking on borrow limits is a more straightforward additive increase by (a conservative estimate of) the amount staked.

5.2.6 Once a borrower has defaulted anywhere in the protocol, they can never again withdraw funds from any part of the protocol.

Status: tested and verified 

Description: A defaulted borrower gets the banned status of the borrowing mutex and is not able to withdraw or claim funds anywhere in the protocol. If a borrower is defaulted and has staked TRU left, the stake can be slashed by the liquidator.

Liquidation and defaults


When a borrower defaults on a loan from a supported pool (for myriad possible reasons – to clarify, this is an OR over default reasons, not an AND), their address is banned from the protocol. A borrower gets banned if he defaults on fixed term loan (line 208) or line of credit (line 535).

```
198 | function enterDefault() external onlyWithdrawn {
199 |     uint256 _unpaidDebt = unpaidDebt();
200 |     require(_unpaidDebt > 0, "FixedTermLoan: Loan must not be fully repaid");
201 |     require(start.add(term).add(creditOracle.gracePeriod()) <= block.timestamp,
202 |         status = Status.Defaulted;
203 |
204 |     debtToken = loanFactory.createDebtToken(pool, borrower, _unpaidDebt);
205 |     debtToken.safeApprove(address(pool), _unpaidDebt);
206 |     pool.addDebt(debtToken, _unpaidDebt);
207 |
208 |     borrowingMutex.ban(borrower);
209 |
210 |     emit Defaulted(debtToken, _unpaidDebt);
211 | }
535 |     borrowingMutex.ban(borrower);
51 | function ban(address borrower) external override onlyLocker(borrower) {
52 |     locker[borrower] = BANNED;
53 |     emit BorrowerBanned(borrower);
54 | }
```

All TRU they may have staked gets slashed, and any unpaid debt gets recorded in a DebtToken owned by the lending pools, that can be brought to legal arbitration. ERC20 funds held by SAFU are used to purchase the DebtTokens from the lending pool, and any remaining debt after liquidation gets recorded in DeficiencyTokens held by the lending pool.

```
131 function compensate(address borrower) external onlyOwner {
132     IDebtToken[] memory debts = loanFactory.debtTokens(borrower);
133
134     for (uint256 i = 0; i < debts.length; i++) {
135         require(debts[i].isLiquidated(), "SAFU: Debt not liquidated yet");
136         ITrueFiPool2 pool = ITrueFiPool2(debts[i].pool());
137         IERC20 token = IERC20(pool.token());
138
139         pool.liquidateDebt(debts[i]);
140         uint256 owedToPool = debts[i].debt().mul(tokenBalance(debts[i])).div(debts[i].totalSupply());
141         uint256 safuTokenBalance = tokenBalance(token);
142         uint256 toTransfer = owedToPool;
143         DeficiencyToken _deficiencyToken;
144         uint256 deficit;
145
146         if (owedToPool > safuTokenBalance) {
147             deficit = owedToPool.sub(safuTokenBalance);
148             toTransfer = safuTokenBalance;
149             _deficiencyToken = new DeficiencyToken(debts[i], deficit);
150             _deficiencyToken[debts[i]] = _deficiencyToken;
151             poolDeficit[address(pool)] = poolDeficit[address(pool)].add(deficit);
152         }
153         token.safeTransfer(address(pool), toTransfer);
154         emit Liquidated(debts[i], toTransfer, _deficiencyToken, deficit);
155     }
156 }
```

5.2.7 TrueFi Protocol cannot be effected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzx

Status: tested and verified 

Description:



1. Celsius :

Celsius network was effected by the BadgerDAO hack in december 2021. This hack was a front-end attack and the smart contracts were untouched. A hacker was able steal an active API key of cloudflare, a content delivery network. With the help of the API key, the attacker injected malicious front-end code where the user was ask to approve a malicious contract address to spend tokens. The approved contract drained all approved tokens out of the users wallet.

This hack can potentially happen to other Dapps like TrueFi when lenders are ask for approval on lending. Users commonly do not double check the approved address and just allow the spending of tokens. To avoid this scenario, it is important to keep the front-end as safe as the contracts. It is recommended to update API keys and access keys like SSH keys regularly and keep them safe.

2. Cream Finance:

In the cream hack in October 2021 \$130m assets of the lending platform were stolen. The smart contracts have been outplayed with a flash loan attack. A borrower flash borrowed funds from Maker and borrowed loan tokens from cream vault multiple times. He manipulated the prices by burning the tokens.

This attack cannot happen to the TrueFi protocol because multiple borrow transaction for one account are disabled. Additionally, the burning of tokens to reduce the total supply and thus manipulate the price is not possible.

3. bzX:

On the latest bzX hack, the private key of the deploying account was compromised by an attacker. An employer stored the mnemonic of the deployer wallet locally on his machine and received an email with a word document. The file ran a malicious macro and stole the mnemonic. With the privilege and funds of the deployer wallet, the attacker could drain out funds from the protocol by using the protocols tokens.

This attack could potentially happen to the TrueFi platform as well. The owner address could set the address of the agencies and borrow funds from the pool with the malicious address. To prevent this scenario, it is recommended to have a security guideline that ensures the safe storage of private keys. Keys should never be stored on a machine that is uses for network interactions such as customer requests. It is recommended to store the privileged private key securely offline. Other options can be multisig Wallets, such as Gnosis Safe.

6. Executive Summary

Three (3) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the December 21, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, no critical issues were found, after the manual and automated security testing and the claim have been successfully verified.

Considering the complexity of TrueFi protocol, the code quality that the TrustToken Team has presented, was exceptional. The inline comments have been helpful to understand the context and the unit tests covered already lots of possible test cases.