Security Assessment

TrueFi

Dec 2nd, 2021

Table of Contents

Summary

This report has been prepared for TrueFi to discover issues and vulnerabilities in the source code of the TrueFi project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	TrueFi
Platform	Ethereum
Language	Solidity
Codebase	• https://github.com/trusttoken/smart-contracts
Commit	• adb9d2faee3f991ce9df8a94ebb318fc6c500748

Audit Summary

Delivery Date	Dec 02, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
Major	8	1	6	0	1	0
Medium	1	0	1	0	0	0
Minor	8	0	3	0	1	4
Informational	2	0	0	0	0	2
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
BMF	contracts/truefi2/BorrowingMutex.sol	60f6af6e5ad26479fb0512125ca471a611df2c1c63bd996ca1e66a075fe64eca
DTF	contracts/truefi2/DebtToken.sol	5a6c3ba83a0d8e85a62f77ff43aa5271c9a9b4e397b068729ad81b6c0a5452de
FTL	contracts/truefi2/FixedTermLoanAgency.sol	68494b81de677431c534790dccfbe4768caa83a401f18a02efaae950cbfa85cf
LFF	contracts/truefi2/LoanFactory2.sol	60655e7062c6073644bb5481126f979f46fb1486a78ad127157464f0233e5d74
TCA	contracts/truefi2/TrueCreditAgency.sol	12c44ac9c20b64a8eabcba9fdd745d4916aec72e73f3466897a5aaebfb04f6f4
TRA	contracts/truefi2/TrueRateAdjuster.sol	be362cf1b6efecdfa45b527d431e5ed5930a792e05ec98b23a513a1f4ba6e3e5
DTC	test/truefi2/lines-of-credit/DebtToken.test.ts	fd1ee7f7dfbf574c902ece7574c0386403d1693136cab445ab817b54253ee854
TCF	test/truefi2/lines-of-credit/TrueCreditAgency.test.ts	098d8f886d2a4fe6e8cf7d8c02e8a957c0869f970509f0f4392ae2e2c3a317a8
TRF	test/truefi2/lines-of-credit/TrueRateAdjuster.test.ts	16ca297d9d638a0e3686a16510d51d55d6314707f4444a8e215b64b1fddd084a
BMC	test/truefi2/BorrowingMutex.test.ts	6f2d02357b86038b0d4e04587e7940d3249873a090e3a6f8dca966094c8c9224
FTA	test/truefi2/FixedTermLoanAgency.test.ts	44c58d431dbcc2335a506ffcac25211173acecf9652f255b3c3f37857461aee6
LFC	test/truefi2/LoanFactory2.test.ts	d46228b8d0d6d38783ca93c106be6e804ee9fb368ebbe72ec45a9b0196a1707f

Review Notes

External Dependencies

There are a few depending injection contracts or addresses in the current project:

- `pool` for the contract `DebtToken`.
- `stakingPool`, `poolFactory`, `_1inch`, `creditOracle`, `rateAdjuster`, `borrowingMutex`, `loanFactory`, and `pool` for the contract `FixedTermLoanAgency`.
- `poolFactory`, `ftlAgency`, `rateAdjuster`, `creditOracle`, `borrowingMutex`, `creditAgency`, and `pool` for the contract `LoanFactory2`.
- `creditOracle`, `rateAdjuster`, `borrowingMutex`, `poolFactory`, `loanFactory`,`pool` for the contract `TrueCreditAgency`.
- `poolFactory`, `pool`, `baseRateOracle` for the contract `TrueRateAdjuster`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privilledged Functions

To set up the project correctly, improve overall project quality and preserve upgradability, the following roles are adopted in the codebase.

In the contract `BorrowingMutex`, the role `owner` has the authority over the following function:

- `BorrowingMutex.allowLocker()` to determine if an account can set the locker of a borrower or not.

In the contract `DebtToken`, the role `liquidator` has the authority over the following function:

- `DebtToken.liquidate()`to liquidate the loan if it has defaulted.

In the contract `FixedTermLoanAgency`, the role `owner` has the authority over the following functions:

- `FixedTermLoanAgency.setCreditOracle()` to set a new `creditOracle` address;
- `FixedTermLoanAgency.setBorrowingMutex()` to set a new `borrowingMutex` address;
- `FixedTermLoanAgency.setMaxLoanTerm()` to set a new maximum loan term;
- `FixedTermLoanAgency.setLongTermLoanThreshold()` to set a new long term loan threshold;
- `FixedTermLoanAgency.setLongTermLoanScoreThreshold()` to set a new long term loan credit score threshold;
- `FixedTermLoanAgency.setLoansLimit()` to set a new loans limit;
- `FixedTermLoanAgency.setFeePool()` to set new fee token and pool;

- `FixedTermLoanAgency.setFee()` to set a new loan interest fee that goes to the stakers;
- `FixedTermLoanAgency.allowBorrower()` to allow a new borrower;
- `FixedTermLoanAgency.blockBorrower()` to block a borrower;
- `FixedTermLoanAgency.reclaim()` to reclaim from a defaulted loan.

In the contract `FixedTermLoanAgency`, the role borrower (`isBorrowerAllowed[borrower] == true`) has the authority over the following function:

- `FixedTermLoanAgency.fund()` to create and fund a loan via `LoanFactory` for a pool supported by `PoolFactory`.

In the contract `FixedTermLoanAgency`, the role supported pool (`poolFactory.isSupportedPool(supportedPool) == true`) has the authority over the following function:

- `FixedTermLoanAgency.transferAllLoanTokens()` to allow pool to transfer all `LoanToken`s to the SAFU in case of liquidation.

In the contract `LoanFactory2`, the role `ftlAgency` has the authority over the following function:

- `LoanFactory2.createFTLALoanToken()` to create a new loan token contract.

In the contract `LoanFactory2`, the role `creditAgency` has the authority over the following function:

- `LoanFactory2.createDebtToken()` to create a new debt token contract.

In the contract `LoanFactory2`, the role `admin` has the authority over the following functions:

- `LoanFactory2.setCreditOracle()` to set a new `creditOracle` address;
- `LoanFactory2.setRateAdjuster()` to set a new `rateAdjuster` address;
- `LoanFactory2.setBorrowingMutex()` to set a new `borrowingMutex` address;
- `LoanFactory2.setLoanTokenImplementation()` to set a new `LoanToken` implementation;
- `LoanFactory2.setCreditAgency()` to set a new `creditAgency` address;
- `LoanFactory2.setLender()` to set a new lender address;
- `LoanFactory2.setDebtTokenImplementation()` to set a new `DebtToken` implementation;
- `LoanFactory2.setFixedTermLoanAgency()` to set a new `ftlAgency` address.

In the contract `TrueRateAdjuster`, the role `owner` has the authority over the following functions:

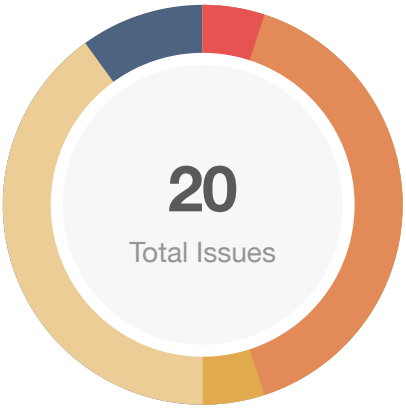
- `TrueRateAdjuster.setRiskPremium()` to set risk premium to `newRate`;
- `TrueRateAdjuster.setCreditScoreRateConfig()` to update credit score rate configuration;
- `TrueRateAdjuster.setUtilizationRateConfig()` to update utilization rate configuration;

- `TrueRateAdjuster.setBaseRateOracle()` to update base rate oracle;
- `TrueRateAdjuster.setFixedTermLoanAdjustmentCoefficient()` to update fixed term loan adjustment coefficient;
- `TrueRateAdjuster.setBorrowLimitConfig()` to update borrow limit configuration.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

[**TrueFi Team**]: There's separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

Findings



Critical	**1** (5.00%)
Major	**8** (40.00%)
Medium	**1** (5.00%)
Minor	**8** (40.00%)
Informational	**2** (10.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Incompatibility With Deflationary Token	Logical Issue	Minor	Declined
GLOBAL-02	Potential Initialization By Frontrunner	Logical Issue	Minor	Declined
BMF-01	Centralization Risk	**Centralization / Privilege**	**Major**	Declined
DTC-01	Missing Test Case	Logical Issue	Minor	Partially Resolved
DTF-01	Centralization Risk	**Centralization / Privilege**	**Major**	Declined
DTF-02	Check-Effect-Interaction Pattern Violation	Logical Issue	Minor	Declined
DTF-03	Logic of Redemption	Logical Issue	Informational	Resolved
FTA-01	Missing Test Case	Logical Issue	Minor	Resolved
FTL-01	Centralization Risk	**Centralization / Privilege**	**Major**	Declined
FTL-02	Potential Integer Overflow	Mathematical Operations	Major	Pending
FTL-03	Potential Reentrancy Attack	Logical Issue	Medium	Declined

ID	Title	Category	Severity	Status
LFF-01	Centralization Risk	**Centralization / Privilege**	**Major**	Declined
LFF-02	Unregistered LoanToken	Logical Issue	Informational	Resolved
TCA-01	Centralization Risk	**Centralization / Privilege**	**Major**	Declined
TCA-02	Logic of Debt Token Distribution	Logical Issue	Critical	Resolved
TCA-03	Potential Integer Overflow	Mathematical Operations	Minor	Resolved
TCF-01	Missing Test Case	Logical Issue	Minor	Resolved
TRA-01	Centralization Risk	**Centralization / Privilege**	**Major**	Declined
TRA-02	Potential Integer Overflow	Mathematical Operations	Major	Partially Resolved
TRF-01	Missing Test Case	Logical Issue	Minor	Resolved

GLOBAL-01 | Incompatibility With Deflationary Token

Category	Severity	Location	Status
Logical Issue	Minor	Global	Declined

Description

When transferring deflationary tokens, the input amount may not equal the received amount due to the charged transaction fees. For example, in the contract `TrueCreditAgency`:

```
746     function _repay(ITrueFiPool2 pool, uint256 amount) internal {
747         pool.token().safeTransferFrom(msg.sender, address(this), amount);
748         pool.token().safeApprove(address(pool), amount);
749         pool.repay(amount);
750     }
```

If a user transfers 100 deflationary tokens (with a 10% transaction fee as an example) in the contract, only 90 tokens actually arrived in the contract. However, the contract needs to transfer 100 tokens to the pool, which causes the contract to lose 10 tokens in such a transaction or revert the transaction due to an insufficient balance.

Recommendation

We advise the client to regulate the set of pool tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[**TrueFi Team**]: Declined. It is not decided yet tokens with fees will be supported. Tether has an option to enable transfer fees so it should be considered.

[**CertiK**]: Although deflationary tokens are not used in the project for now, it is highly recommended to consider this issue when a new token is supported in the system.

GLOBAL-02 | Potential Initialization By Frontrunner

Category	Severity	Location	Status
Logical Issue	Minor	Global	Declined

Description

In the following contracts, the function `initialize()` can be called by anyone to initialize the contracts:

- `BorrowingMutex`
- `DebtToken`
- `FixedTermLoanAgency`
- `LoanFactory2`
- `TrueCreditAgency`
- `TrueRateAdjuster`

Although the project deployer can discard incorrectly initialized contracts, it might still bring errors if the deployment is not properly processed. One of the possible scenarios is described as below:

1. The deployer writes a script to deploy and initialize the contract.
2. The attacker noticed the deployment and initialized the contract before the initialization by the deployer is committed.
3. The deployment script mistakenly ignores the error of initializing the contract (because it has already been initialized) and continues executing other transactions in the script. In this way, the attacker can inject suspicious addresses into the contracts.

Recommendation

We recommend adding proper access control to the function `initialize()` in the aforementioned contracts or checking the status of initialization in the deployment process.

Alleviation

[**TrueFi Team**]: Declined. We could add `onlyProxyOwner` or something to initializers but I don't think it's worth it. The attacker would have to monitor all deployed contracts and know where to call initialize. In case it's done, we just redeploy the contract.

[**CertiK**]: If the team checks the results of initializations and redeploy contracts if they are front-run, this issue will not bring troubles to the project.

BMF-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/trueFi/contracts/truefi2/BorrowingMutex.sol (38edf01): 31	Declined

Description

In the contract `BorrowingMutex`, the role `owner` has the authority over the following function:

- `BorrowingMutex.allowLocker()` to determine if an account can set the locker of a borrower or not.

If an account is set to `canLock`, it will be able to lock any unlocked borrower with any account as its locker.

Any compromise to the `owner` account may allow the hacker to take advantage of this and manipulate the project.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**TrueFi Team**]: Declined. There's separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

[**CertiK**]: Multi-signature wallet is one of the solutions to reduce the risk. It is still recommended to consider Timelock and DAO for fully decentralized governance of the project.

DTC-01 | Missing Test Case

Category	Severity	Location	Status
Logical Issue	Minor	projects/trueFi/test/truefi2/lines-of-credit/DebtToken.test.ts (38edf01): 11	Partially Resolved

Description

The following functions in the contract `DebtToken` are not fully covered by unit tests:

- The function `DebtToken.redeem()` has the require statement `require(status >= Status.Defaulted, "DebtToken: The debt has not defaulted yet");`. There is no test for the case when `status >= Status.Defaulted`.
- There is no test for the function `DebtToken.balance()`.

Recommendation

We recommend adding tests for the aforementioned scenarios.

Alleviation

The TrueFi team heeded our advice and partially resolved this issue by adding tests for `DebtToken.balance()` in the commit `1f7fd61f21e50a4819bfe719fa0a68a9ae035a97`.

DTF-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/trueFi/contracts/truefi2/DebtToken.sol (38edf01): 78	⊗ Declined

Description

In the contract `DebtToken`, the role `liquidator` has the authority over the following function:

- `DebtToken.liquidate()` to liquidate the loan if it has defaulted.

Any compromise to the `liquidator` account may allow the hacker to take advantage of this and manipulate the project.

Recommendation

We advise the client to carefully manage the `liquidator` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**TrueFi Team**]: Declined. There's separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

[**CertiK**]: Multi-signature wallet is one of the solutions to reduce the risk. It is still recommended to consider Timelock and DAO for fully decentralized governance of the project.

DTF-02 | Check-Effect-Interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	projects/trueFi/contracts/truefi2/DebtToken.sol (38edf01): 59	Declined

Description

The Solidity documentation suggests that a smart contract should follow the `Checks-Effects-Interactions` pattern. However, the function `DebtToken.redeem()` violates the `Checks-Effects-Interactions` pattern by having an external call (Interaction) before an event emission (Effect).

Recommendation

We advise the client to adopt the `Checks-Effects-Interactions` pattern in the aforementioned function by, for example, emitting the event before processing the external call, or apply the non-reentrancy guardian to the function.

Reference: https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Alleviation

[**TrueFi Team**]: Acknowledged and declined. This should be done, but to match the code style in the rest of the repo we will push this task to a later refactoring. Event emission is mostly considered a side effect and not consumed on-chain, so we consider the reentrancy risk to be low at the moment.

[**CertiK**]: The auditors agree that if the event does not play a key role in the system the risk of reentrancy would be low.

DTF-03 | Logic of Redemption

Category	Severity	Location	Status
Logical Issue	Informational	projects/trueFi/contracts/truefi2/DebtToken.sol (38edf01): 65	Resolved

Description

The function `DebtToken.redeem()` burns debt token from the caller's account and transfers `pool.token()` to the caller. When `amount == totalSupply()` and `repaid() > debt`, the amount to transfer will be set to the balance of the contract:

```
64          uint256 amountToReturn = _amount;
65          if (_amount == totalSupply() && repaid() > debt) {
66              amountToReturn = _balance();
67          }
```

However, based on our understanding, the condition `repaid() > debt` is not necessary:

1. `redeemed == debt - totalSuppy()` before all debt tokens are burnt, because (1) `totalSuppy() == debt` when the contract is initialized; (2) the value of `redeemed` is updated only when `_burn()` is triggered (L68~69); (3) their changes are always equivalent unless `_amount == totalSupply()` which means all debt tokens will be burnt.
2. When `_amount == totalSupply()`, `repaid() == _balance() + redeemed = _balance() + debt - totalSuppy() = _balance() + debt - _amount`.
3. Because `_amount <= _balance()` (L62), `repaid() == _balance() + debt - _amount >= _amount + debt - _amount == debt`.
4. Therefore, when `_amount == totalSupply()`, `repaid() >= debt`. And `repaid() == debt` only when `_amount == _balance()`. The following code should have exactly the same results:

```
64          uint256 amountToReturn = _amount;
65          if (_amount == totalSupply()) {
66              amountToReturn = _balance();
67          }
```

To make sure we understand the logic of redemption correctly, we would like to check with the TrueFi team what would be the purpose of introducing the condition `repaid() > debt`.

Alleviation

The TrueFi team confirmed the aforementioned condition is unnecessary and removed it in the commit `b2e10db4d5eabfd4a69fb7121567baff54ab8d21`.

FTA-01 | Missing Test Case

Category	Severity	Location	Status
Logical Issue	Minor	projects/trueFi/test/truefi2/FixedTermLoanAgency.test.ts (38edf01): 42	Resolved

Description

For the contract `FixedTermLoanAgency`, the following scenario is not tested:

- Multiple loans from different borrowers.

The following function is not covered by test:

- `FixedTermLoanAgency.transferAllLoanTokens()`.

Recommendation

We recommend adding tests for the aforementioned scenarios.

Alleviation

[**TrueFi Team**]: The code is obsolete.

[**CertiK**]: The function `FixedTermLoanAgency.transferAllLoanTokens()` has been removed.

FTL-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/trueFi/contracts/truefi2/FixedTermLoanAgency.sol (38edf01): 215, 224, 233, 242, 251, 260, 270, 280, 286, 291, 378, 323, 528	Declined

Description

In the contract `FixedTermLoanAgency`, the role `owner` has the authority over the following functions:

- `FixedTermLoanAgency.setCreditOracle()` to set a new `creditOracle` address;
- `FixedTermLoanAgency.setBorrowingMutex()` to set a new `borrowingMutex` address;
- `FixedTermLoanAgency.setMaxLoanTerm()` to set a new maximum loan term;
- `FixedTermLoanAgency.setLongTermLoanThreshold()` to set a new long term loan threshold;
- `FixedTermLoanAgency.setLongTermLoanScoreThreshold()` to set a new long term loan credit score threshold;
- `FixedTermLoanAgency.setLoansLimit()` to set a new loans limit;
- `FixedTermLoanAgency.setFeePool()` to set new fee token and pool;
- `FixedTermLoanAgency.setFee()` to set a new loan interest fee that goes to the stakers;
- `FixedTermLoanAgency.allowBorrower()` to allow a new borrower;
- `FixedTermLoanAgency.blockBorrower()` to block a borrower;
- `FixedTermLoanAgency.reclaim()` to reclaim from a defaulted loan.

The role borrower (`isBorrowerAllowed[borrower] == true`) has the authority over the following function:

- `FixedTermLoanAgency.fund()` to create and fund a loan via `LoanFactory` for a pool supported by `PoolFactory`.

The role supported pool (`poolFactory.isSupportedPool(supportedPool) == true`) has the authority over the following function:

- `FixedTermLoanAgency.transferAllLoanTokens()` to allow pool to transfer all `LoanToken`s to the SAFU in case of liquidation.

Any compromise to the `owner` account and supported pools may allow the hacker to manipulate the project through these functions.

Recommendation

We advise the client to carefully manage the `owner` account's private key and supported pools to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**TrueFi Team**]: Declined. There’s separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

[**CertiK**]: Multi-signature wallet is one of the solutions to reduce the risk. It is still recommended to consider Timelock and DAO for fully decentralized governance of the project.

FTL-02 | Potential Integer Overflow

Category	Severity	Location	Status
Mathematical Operations	Major	projects/trueFi/contracts/truefi2/FixedTermLoanAgency.sol (38edf01): 412, 416, 420, 418, 497	Pending

Description

The maximum value of `uint8` is 255 while The maximum value of `uint256` is `2**256 - 1`. In the mathematical operations

```
412         uint256 resultPrecision = uint256(10)**decimals;
```

```
418             uint256 poolPrecision =
uint256(10)**ITrueFiPool2WithDecimals(address(pool)).decimals();
```

```
418         uint256 expectedDiff =
pool.oracle().tokenToUsd(feeAmount).mul(10**feeToken.decimals()).div(1 ether);
```

the following calculations have the risk of integer overflow:

- `uint256(10)**decimals`
- `uint256(10)**ITrueFiPool2WithDecimals(address(pool)).decimals()`
- `10**feeToken.decimals()`

In the `for` loop in L416 and L420, the data type of indexes is `uint8`:

```
        for (uint8 i = 0; i < pools.length; i++) {
            ...
            for (uint8 j = 0; j < _loans.length; j++) {
                ...
            }
        }
```

It is possible that `pools.length` or `j < _loans.length` is larger than 255, in which case integer overflow might happen when updating indexes by `i++` or `j++`.

Recommendation

We recommend checking integer overflows for all addition calculations and setting restrictions for the powers in power calculations, or updating the Solidity version to 0.8.x. In Solidity 0.8.x, the transaction will be reverted if integer overflow happens.

Alleviation

The TrueFi team heeded our advice and resolved this issue in the PR 1093.

FTL-03 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Medium	projects/trueFi/contracts/truefi2/FixedTermLoanAgency.sol (38edf01): 323, 378	Declined

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

The functions `FixedTermLoanAgency.fund()` and `FixedTermLoanAgency.reclaim()` have state updates or event emissions after external calls and thus are vulnerable to reentrancy attacks.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[**TrueFi Team**]: Declined. This should be done, but to match the code style in the rest of the repo we will push this task to a later refactoring. Event emission is mostly considered a side effect and not consumed on-chain, so we consider the reentrancy risk to be low at the moment.

[**CertiK**]: When there is more than one external call, it is possible that the second external call makes some state update. In this case, the first external call might affect the state update in the second one.

LFF-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/trueFi/contracts/truefi2/LoanFactory2.sol (38edf01): 169, 201, 216, 222, 228, 234, 240, 246, 252, 258	Declined

Description

In the contract `LoanFactory2`, the role `ftlAgency` has the authority over the following function:

- `LoanFactory2.createFTLALoanToken()` to create a new loan token contract.

The role `creditAgency` has the authority over the following function:

- `LoanFactory2.createDebtToken()` to create a new debt token contract.

The role `admin` has the authority over the following functions:

- `LoanFactory2.setCreditOracle()` to set a new `creditOracle` address;
- `LoanFactory2.setRateAdjuster()` to set a new `rateAdjuster` address;
- `LoanFactory2.setBorrowingMutex()` to set a new `borrowingMutex` address;
- `LoanFactory2.setLoanTokenImplementation()` to set a new `LoanToken` implementation;
- `LoanFactory2.setCreditAgency()` to set a new `creditAgency` address;
- `LoanFactory2.setLender()` to set a new lender address;
- `LoanFactory2.setDebtTokenImplementation()` to set a new `DebtToken` implementation;
- `LoanFactory2.setFixedTermLoanAgency()` to set a new `ftlAgency` address.

Any compromise to the role `admin`, `ftlAgency` or `creditAgency` may allow the hacker to manipulate the project through these functions.

Recommendation

We advise the client to carefully manage the `admin` account's private key, the contracts `ftlAgency` and `creditAgency` (we assume these two roles are set to the `FixedTermLoanAgency` contract and the `TrueCreditAgency` contract respectively) to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**TrueFi Team**]: Declined. There's separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

[**CertiK**]: Multi-signature wallet is one of the solutions to reduce the risk. It is still recommended to consider Timelock and DAO for fully decentralized governance of the project.

LFF-02 | Unregistered LoanToken

Category	Severity	Location	Status
Logical Issue	Informational	projects/trueFi/contracts/truefi2/LoanFactory2.sol (38edf01): 154	Resolved

Description

The contract `FixedTermLoanAgency` can call `LoanFactory2.createFTLALoanToken()` to create a new `LoanToken` contract and add it to the `poolLoans` list which registers the loan for the pool.

Meanwhile the function `LoanFactory2.createLoanToken()` can be triggered without access restrictions to create a new `LoanToken` contract, which is not registered for the pool.

Although the unregistered `LoanToken` contract does receive funds from the pool, unnecessary `LoanToken` creation might cause confusion in the market.

Recommendation

We recommend removing the function `LoanFactory2.createLoanToken()` if it is unnecessary.

Alleviation

The TrueFi team has removed the function `LoanFactory2.createLoanToken()` in the commit `9138a94724105a8d8145a075bf539f79f79c3380`.

TCA-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/trueFi/contracts/truefi2/TrueCreditAgency.sol (38edf01): 193, 341, 200, 207, 214, 220, 226, 341, 427	Declined

Description

In the contract `TrueCreditAgency`, the role `owner` has the authority over the following functions:

- `TrueCreditAgency.setRateAdjuster()` to set a new `rateAdjuster` address and update pool state;
- `TrueCreditAgency.setPoolFactory()` to set a new `poolFactory` address and update pool state;
- `TrueCreditAgency.setLoanFactory()` to set a new `loanFactory` address and update pool state;
- `TrueCreditAgency.setInterestRepaymentPeriod()` to set a new interest repayment period;
- `TrueCreditAgency.setMinCreditScore()` to set a new minimum credit score;
- `TrueCreditAgency.allowBorrower()` to set whitelist a borrower;
- `TrueCreditAgency.enterDefault()` to enter default for a certain borrower's lines of credit.

The role allowed borrower (`isBorrowerAllowed[borrower] == true`) has the authority over the following function:

- `TrueCreditAgency.borrow()` to borrow tokens from pools using lines of credit.

Any compromise to the `owner` or allowed borrower account may allow the hacker to take advantage of this and manipulate the project through these functions.

Recommendation

We advise the client to carefully manage the `owner` and allowed borrower accounts' private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**TrueFi Team**]: Declined. There's separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

[**CertiK**]: Multi-signature wallet is one of the solutions to reduce the risk. It is still recommended to consider Timelock and DAO for fully decentralized governance of the project.

TCA-02 | Logic of Debt Token Distribution

Category	Severity	Location	Status
Logical Issue	Critical	projects/trueFi/contracts/truefi2/TrueCreditAgency.sol (38edf01)	Resolved

Description

In the contract `TrueFiPool2`, there is a function `TrueFiPool2.addDebt()` transferring DebtToken from `creditAgency` to the pool:

```
632     function addDebt(IDebtToken debtToken, uint256 amount) external {
633         require(msg.sender == address(creditAgency), "TruePool: Only
TrueCreditAgency can add debtTokens");
634         debtValue = debtValue.add(amount);
635         debtToken.safeTransferFrom(msg.sender, address(this), amount);
636 
637         emit DebtAdded(debtToken, amount);
638     }
```

This function can only be called by `creditAgency`. We assume `creditAgency` refers to the contract `TrueCreditAgency` because it has the interface `ITrueCreditAgency`. However, `TrueFiPool2.addDebt()` is never called within the contract `TrueCreditAgency`.

We would like to check with the TrueFi team if there is any missing logic for the debt token distribution.

Alleviation

The TrueFi team has resolved this issue in the commit `90acca34b4957de02b8e3838fe78d7f0a4023b96`.

TCA-03 | Potential Integer Overflow

Category	Severity	Location	Status
Mathematical Operations	Minor	projects/trueFi/contracts/truefi2/TrueCreditAgency.sol (38edf01): 670	Resolved

Description

In the borrower counting

```
670         bucket.borrowersCount = bucket.borrowersCount + 1;
```

the variable `bucket.borrowersCount` is uint16, whose maximum is 65535. We would like check with the TrueFi team if it is possible that the number of borrower would be greater than 65535.

Alleviation

The TrueFi team heeded our advice and resolved this issue in the PR 1092.

TCF-01 | Missing Test Case

Category	Severity	Location	Status
Logical Issue	Minor	projects/trueFi/test/truefi2/lines-of-credit/TrueCreditAgency.test.ts (38edf01): 36	Resolved

Description

For the contract `TrueCreditAgency`, the following functions are not covered by tests:

- `TrueCreditAgency.setRateAdjuster()`
- `TrueCreditAgency.pokeAll()`

Recommendation

We recommend adding tests for the aforementioned scenarios.

Alleviation

The TrueFi team heeded our advice and resolved this issue in the PR 1085.

TRA-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Major**	projects/trueFi/contracts/truefi2/TrueRateAdjuster.sol (38edf01): 130, 135, 140, 146, 152, 164	Declined

Description

In the contract `TrueRateAdjuster`, the role `owner` has the authority over the following functions:

- `TrueRateAdjuster.setRiskPremium()` to set risk premium to `newRate`;
- `TrueRateAdjuster.setCreditScoreRateConfig()` to update credit score rate configuration;
- `TrueRateAdjuster.setUtilizationRateConfig()` to update utilization rate configuration;
- `TrueRateAdjuster.setBaseRateOracle()` to update base rate oracle;
- `TrueRateAdjuster.setFixedTermLoanAdjustmentCoefficient()` to update fixed term loan adjustment coefficient;
- `TrueRateAdjuster.setBorrowLimitConfig()` to update borrow limit configuration.

Any compromise to the `owner` account may allow the hacker to take advantage of this and manipulate the project through these functions.

Recommendation

We advise the client to carefully manage the `owner` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[**TrueFi Team**]: Declined. There’s separate ongoing work on progressive decentralization. Special roles are currently controlled by Gnosis Safe multisigs that will slowly be granted to the community.

[**CertiK**]: Multi-signature wallet is one of the solutions to reduce the risk. It is still recommended to consider Timelock and DAO for fully decentralized governance of the project.

TRA-02 | Potential Integer Overflow

Category	Severity	Location	Status
Mathematical Operations	Major	projects/trueFi/contracts/truefi2/TrueRateAdjuster.sol (38edf01): 230, 246	Partially Resolved

Description

The maximum value of `uint16` is 65535 while The maximum value of `uint256` is `2**256-1`. In the mathematical operations

```
230         return
min(coefficient.mul(uint256(MAX_CREDIT_SCORE)**power).div(uint256(score)**power).sub(coef
ficient), MAX_RATE_CAP);
```

```
246         return
min(coefficient.mul(1e4**power).div(liquidRatio**power).sub(coefficient), MAX_RATE_CAP);
```

the following calculations have the risk of integer overflow:

- `uint256(MAX_CREDIT_SCORE)**power`
- `uint256(score)**power`
- `coefficient.mul(1e4**power)`
- `liquidRatio**power`

Recommendation

We recommend setting restrictions for the state variable `power` or updating the Solidity version to 0.8.x. In Solidity 0.8.x, the transaction will be reverted if integer overflow happens.

Alleviation

[**TrueFi Team**]: It is updated in the PR 1087.

[**CertiK**]: In PR 1087, one of the comments mentioned it is unlikely to happen that `power` is greater or equal to 19. While the auditors agree that the integer overflow only happens when some values are extremely large, the possibilities of integer overflow cannot be excluded if there is no input validation for relevant calculations.

TRF-01 | Missing Test Case

Category	Severity	Location	Status
Logical Issue	Minor	projects/trueFi/test/truefi2/lines-of-credit/TrueRateAdjuster.test.ts (38edf01): 27	Resolved

Description

For the contract `TrueRateAdjuster`, the following function has a missing test case:

- `TrueRateAdjuster.borrowLimit()`: when the credit score is below the minimum required score.

Recommendation

We recommend adding tests for the aforementioned scenarios.

Alleviation

The TrueFi team heeded our advice and resolved this issue in the PR 1084.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.