SOLIDIFIED

Audit Report for TrueFi - August 10, 2022

Summary

Audit Report prepared by Solidified covering the Truefi Helium version Ethereum smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code.
The debrief on 7 April 2022.
A second round of auditing concluded on 7 June 2022

Audited Files

The source code has been supplied in the form of the following GitHub repository:

https://github.com/trusttoken/contracts-helium

Commit number: `2226a66dcd737217a7eaa8eb29c7bda952a53b2a`

The scope of the audit was limited to the following files:

```
- BasePortfolio.sol
- BasePortfolioFactory.sol
- FlexiblePortfolio.sol
- FlexiblePortfolioFactory.sol
- FixedInterestOnlyLoans.sol
- AutomatedLineOfCredit.sol
- AutomatedLineOfCreditFactory.sol
- governance/DaoGovernor.sol
- governance/DaoToken.sol
- governance/StkTruToken.sol
- governance/VoteToken.sol
- ProtocolConfig.sol
- strategies/DepositStrategy.sol
- strategies/FIOLValuationStrategy.sol
- strategies/LiquidValuationStrategy.sol
- strategies/MultiInstrumentValuationStrategy.sol
- strategies/NonUsOnlyDeposit.sol
- strategies/TransferStrategy.sol
- strategies/WhitelistDeposit.sol
- strategies/WithdrawStrategy.sol
- governance/common/ERC20.sol
- governance/common/ProxyStorage.sol
- governance/common/StkClaimableContract.sol
```

```
- BasePortfolio.sol
- BasePortfolioFactory.sol
```

- FlexiblePortfolio.sol
- FlexiblePortfolioFactory.sol
- FixedInterestOnlyLoans.sol
- AutomatedLineOfCredit.sol
- AutomatedLineOfCreditFactory.sol
- governance/DaoGovernor.sol
- governance/DaoToken.sol
- governance/StkTruToken.sol
- governance/VoteToken.sol
- access/Manageable.sol
- access/InitializableManageable.sol
- ProtocolConfig.sol
- strategies/DepositStrategy.sol
- strategies/FIOLValuationStrategy.sol
- strategies/LiquidValuationStrategy.sol
- strategies/MultiInstrumentValuationStrategy.sol
- strategies/NonUsOnlyDeposit.sol
- strategies/TransferStrategy.sol
- strategies/WhitelistDeposit.sol
- strategies/WithdrawStrategy.sol
- governance/common/ERC20.sol
- governance/common/ProxyStorage.sol
- governance/common/StkClaimableContract.sol

Intended Behavior

The smart contracts implement an uncollateralized lending protocol that allows asset managers to create lending pools in which liquidity providers deposit funds. Asset managers can also issue loans to borrowers, which should be repaid by a certain date.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than in a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the TrueFi contracts contain 1 critical issue, 2 major issues, 4 minor issues in addition to 9 informational notes.

In addition, one warning has been added.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Portfolio managers can access and remove all funds	Warning	-
2	BasePortfolio.soll: Unprotected functions **Injected bug during audit benchmarking**	Critical	Non-issue
3	AutomatedLineOfCredit.sol: Unprotected functions	Major	Resolved
4	FixedInterestOnlyLoans.sol: Function updateInstrument() allows the loan owner to set gracePeriod to a value lower than what the loan recipient had accepted	Major	Resolved
5	FlexiblePortfolio.sol: Contract can receive ERC-721 tokens but does not handle them	Minor	Acknowledged
6	AutomatedLineOfCredit.sol: Unpaid interest may be missed in some cases	Minor	Resolved
7	FlexiblePortfolio.sol: Valuation strategy hook is not called on instrument update	Minor	Resolved
8	MultiInstrumentValuationStrategy.sol: Function addStrategy() does not validate the provided instrument	Minor	Resolved
9	FlexiblePortfolio.sol: Instruments can be funded before having been added	Note	Resolved

10	Absence of Zero Address validation	Note	-
11	BasePortfolio.sol: Function withdraw() failures revert without meaningful error messages	Note	Resolved
12	AutomatedLineOfCredit.sol: Absence of input validation in the setMaxSize() function	Note	-
13	FlexiblePortfolio.sol: Function deposit() allows zero net deposit amounts	Note	-
14	StkTruToken.sol: Redundant constructor	Note	-
15	FlexiblePortfolio.sol: Redundant implementation of function transfer()	Note	-
16	AutomatedLineOfCredit.sol: The check address(this) != borrower should be done in initialize()	Note	-
17	Use of floating pragma	Note	-

Warnings

1. Portfolio managers can access and remove all funds

The portfolio manager role is extremely powerful. A manager can do the following:

- Create loans to any address, including themselves and other addresses owned by themselves.
- Change the fees at will.
- Mark loans as defaulted at will.
- Pause portfolios.

This essentially means that the manager can steal funds at will.

Recommendation
Consider adding the following safeguards:

- Not allowing fees to be modified after portfolio creation or placing bounds on fees
- Add checks on loans being declared defaulted (see issue below)
- Implementing a whitelisting/KYC procedure for borrowers (not just lenders and managers)
- Not allowing managers to issue loans to themselves

Critical Issues

2. BasePortfolio.sol: Unprotected functions

The functions `setTransferStrategy()`, is unprotected and can be called by anyone. This can be exploited by setting a malicious transfer strategy.

Recommendation
Add a modifier to protect the functions from unauthorized calls.

Team Reply
The team has identified this issue as a bug injected for audit benchmarking. The team provided hashes of injected bug descriptions before the audit, and will also publicly reveal them to their community.

Major Issues

3. AutomatedLineOfCredit.sol: Unprotected functions

The functions `repay()` and `repayInFull()` are unprotected and can be called by anyone. .

Recommendation
Add a modifier to protect the functions from unauthorized calls.

4. FixedInterestOnlyLoans.sol: Function updateInstrument() allows the loan owner to set gracePeriod to a value lower than what the loan recipient had accepted

Function `updateInstrument()` should only be able to increase the value of `gracePeriod` after the loan recipient has accepted the loan. This could be used to immediately force a loan to default by changing the grace period.

Recommendation

Enforce that `newGracePeriod > loans[instrumentId].gracePeriod`.

Minor Issues

5. FlexiblePortfolio.sol: Contract can receive ERC-721 tokens but does not handle them

The contract implements the function `onERC721Received()`. However, there is no code that allows for dealing with ERC-721 tokens, nor does it seem to require them. Tokens transferred to the contract may simply get stuck.

Recommendation

Remove the `onERC721Received()` function or implement code to deal with ERC-721 tokens.

Team Reply

“If there are any other ERC 721 tokens than ours in any of the portfolios, we could easily update its implementation (because we deploy it as a proxy) and add a function to handle these tokens. Our own loans represented as ERC 721 tokens being stuck in portfolios is actually an intended behavior.”

6. AutomatedLineOfCredit.sol: Unpaid interest may be missed in some cases

The function `_value()` appears to calculate the value without taking into account unpaid interests, which contradicts the documentation supplied. The documentation states that value should be calculated as follows:

```
liquid_funds_in_ALOC + borrowed_funds + unpaid_interest -
unclaimed_fees
```

Recommendation

Adapt the calculation to match the specification.

7. FlexiblePortfolio.sol: Valuation strategy hook is not called on instrument update

The function `updateInstrument()` fails to call the `valuationStrategy.onInstrumentUpdated()` handler provided for such an occurrence. This may have undesired effects in some instrument and strategy combinations.

Recommendation
Add a call to the valuation strategy hook.

8. MultiInstrumentValuationStrategy.sol: Function addStrategy() does not validate the provided instrument

The documentation of `MultiInstrumentValuationStrategy` states that "*it should not be possible to add unexpected instruments*". However, function `addStrategy()` does not provide any form of validation for the given `instrument` before adding it to the `instruments` array.

Recommendation

Provide validation that conforms to the `MultiInstrumentValuationStrategy` documentation.

Informational Notes

9. `FlexiblePortfolio.sol`: Instruments can be funded before having been added

The function `fundInstrument()` can be called on nonexistent instruments. Whilst this will not result in loss of funds, due to causing a zero amount transfer, some data-structure inconsistency might occur. In addition, off-chain components might get confused by the event emitted.

Recommendation

Add a check to confirm that an instrument has been added to the portfolio before funding it.

10. Absence of Zero Address validation

The contracts `BasePortfolio.sol` and `ProtocolConfig.sol` include functions, i.e., `setPortfolioImplementation()` and `setProtocolAddress()` respectively, that update the state of crucial addresses in the contract but do not include any zero address validations

Recommendation

Consider adding `require()` statements to ensure only valid addresses are passed as arguments.

11. `BasePortfolio.sol`: Function `withdraw()` failures revert without meaningful error messages

Withdrawal failures do not give any meaningful error messages back to the caller. In case they have insufficient `shares` balance for instance, they will only get the obscure "`burn amount exceeds balance`" error, which might not make much sense to them in the current context.

Recommendation

Check if the account does not have enough `shares` to withdraw, then revert if true with a meaningful error message.

12. `AutomatedLineOfCredit.sol`: Absence of input validation in the `setMaxSize()` function

The `setMaxSize()` function doesn't validate the `_maxSize` argument being passed to the function.

Recommendation

Consider checking that the argument is larger than 0 and within a reasonable upper bound.

13. `FlexiblePortfolio.sol`: Function `deposit()` allows zero net deposit amounts

Function `deposit()` allows zero net deposit amounts after `managersPart` and `protocolsPart` have been subtracted from the given `amount`.

Recommendation

Consider checking that `protocolsPart + managersPart < amount` instead of `protocolsPart + managersPart <= amount`.

14. StkTruToken.sol: Redundant constructor

The contract `StkTruToken` is designed to be accessed via the *proxy pattern*, and thus any initializations done in the constructor are redundant as initial values are always written to incorrect contract storage.

Recommendation

Remove the contract constructor to save on deployment gas fees.

15. FlexiblePortfolio.sol: Redundant implementation of function transfer()

The base contract `BasePortfolio` already overrides `_transfer()` to have it revert during `whenNotPaused` is active, which makes that current `transfer()` function implementation redundant.

Recommendation

Remove the redundant implementation to save on deployment gas fees.

16. `AutomatedLineOfCredit.sol`: The check `address(this) != borrower` should be done in `initialize()`

Recommendation

Consider having the `address(this) != borrower` check be done once in `initialize()` instead of multiple times in all of the borrow and repay functions.

17. Use of floating pragma

Contracts should always be deployed with the same compiler version that they have been tested with. Locking the pragma helps ensure that contracts do not accidentally get deployed using an outdated compiler version that might introduce unintended side-effects.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of TrustToken or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors from legal and financial liability.

Oak Security GmbH