Smart Contract Security Audit Report

Contents

1. Executive Summary

On Nov. 11, 2020, the SlowMist security team received the TrueFi team's security audit application for TrueFi, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack

- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

TrueFi is a protocol for creating interest bearing pools with a high APR for liquidity providers. TrueFi includes a utility and rewards mechanisms using TrustTokens (TRU) and rewards participants for maintaining stable, high APRs. Un-collateralized loans and decentralized lending are utilized in order to achieve high interest rates for pools.

Audit version code:

https://github.com/trusttoken/smart-contracts/tree/f34c5737cf80c84362217e65b70297cdcb5a485c/contracts/truefi

Files Out of Audit Scope:

https://github.com/trusttoken/smart-contracts/tree/f34c5737cf80c84362217e65b70297cdcb5a485c/contracts/truefi/distributors/FastTrueDistributor.sol
https://github.com/trusttoken/smart-contracts/tree/f34c5737cf80c84362217e65b70297cdcb5a485c/contracts/truefi/distributors/QuadraticTrueDistributor.sol

https://github.com/trusttoken/smart-contracts/tree/f34c5737cf80c84362217e65b70297cdcb5a485c/contracts/truefi/distributors/SlowTrueDistributor.sol

The documents provided by the TrueFi team are as follows:

https://github.com/trusttoken/truefi-spec

https://github.com/trusttoken/truefi-spec/blob/master/LoanToken%20Lifecycle.png

https://github.com/trusttoken/truefi-spec/blob/master/TrueFi%20Product%20Flows.svg

4. Code Overview

4.1 Contracts Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

ArbitraryDistributor			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
distribute	Public	Can modify state	onlyBeneficiary
empty	Public	Can modify state	onlyOwner

LinearTrueDistributor			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
setFarm	External	Can modify state	onlyOwner
distribute	Public	Can modify state	onlyBeneficiary
empty	Public	Can modify state	onlyOwner

ERC20			
Function Name	Visibility	Mutability	Modifiers
__ERC20_initialize	Internal	Can modify state	initializer

name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
totalSupply	Public	-	-
balanceOf	Public	-	-
transfer	Public	Can modify state	-
allowance	Public	-	-
approve	Public	Can modify state	-
transferFrom	Public	Can modify state	-
increaseAllowance	Public	Can modify state	-
decreaseAllowance	Public	Can modify state	-
_transfer	Internal	Can modify state	-
_mint	Internal	Can modify state	-
_burn	Internal	Can modify state	-
_approve	Internal	Can modify state	-
_setupDecimals	Internal	Can modify state	-
_beforeTokenTransfer	Internal	Can modify state	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
initialize	Internal	Can modify state	initializer
owner	Public	-	-
renounceOwnership	Public	Can modify state	onlyOwner
transferOwnership	Public	Can modify state	onlyOwner

Initializable			
Function Name	Visibility	Mutability	Modifiers
isConstructor	Private	-	-

LoanToken			
Function Name	Visibility	Mutability	Modifiers
isLoanToken	External	-	-
getParameters	External	-	-
value	External	-	-

fund	External	Can modify state	onlyAwaiting
allowTransfer	External	Can modify state	onlyLender
withdraw	External	Can modify state	onlyBorrower onlyFunded
close	External	Can modify state	onlyOngoing
redeem	External	Can modify state	onlyClosed
repay	External	Can modify state	onlyAfterWithdraw
repaid	External	Can modify state	onlyAfterWithdraw
balance	External	-	-
_balance	Internal	-	-
receivedAmount	Public	-	-
interest	Internal	-	-
profit	External	-	-
_transfer	Internal	Can modify state	onlyWhoCanTransfer

TrueFarm			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
stake	External	Can modify state	update
_unstake	Internal	Can modify state	-
_claim	Internal	Can modify state	-
unstake	External	Can modify state	update
claim	External	Can modify state	update
exit	External	Can modify state	update

LoanFactory			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can modify state	initializer
createLoanToken	External	Can modify state	

TrueFiPool			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can modify state	initializer
currencyToken	Public	-	-

yTokenBalance	Public	-	-
poolValue	Public	-	-
ensureEnoughTokensAreAvailable	Internal	Can modify state	-
setJoiningFee	External	Can modify state	onlyOwner
join	External	Can modify state	-
exit	External	Can modify state	nonReentrant
flush	External	Can modify state	onlyOwner
pull	External	Can modify state	onlyOwner
borrow	External	Can modify state	nonReentrant
repay	External	Can modify state	-
collectCrv	External	Can modify state	onlyOwner
collectFees	External	Can modify state	onlyOwner
calcTokenAmount	Public	-	-
calcWithdrawOneCoin	Public	-	-
currencyBalance	Internal	-	-

TrueLender			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
setSizeLimits	External	Can modify state	onlyOwner
setTermLimits	External	Can modify state	onlyOwner
setApyLimits	External	Can modify state	onlyOwner
setVotingPeriod	External	Can modify state	onlyOwner
setParticipationFactor	External	Can modify state	onlyOwner
setRiskAversion	External	Can modify state	onlyOwner
loans	Public	-	-
allow	External	Can modify state	onlyOwner
fund	External	Can modify state	onlyAllowedBorrowers
value	External	-	-
reclaim	External	Can modify state	onlyOwner
distribute	External	Can modify state	onlyPool
loanIsAttractiveEnough	Public	-	-
votingLastedLongEnough	Public	-	-
loanSizeWithinBounds	Public	-	-
loanTermWithinBounds	Public	-	-
votesThresholdReached	Public	-	-

loanIsCredible	Public	-	-

TrueRatingAgency.sol			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
setLossFactor	External	Can modify state	onlyOwner
setBurnFactor	External	Can modify state	onlyOwner
getNoVote	Public	-	-
getYesVote	Public	-	-
getTotalNoVotes	Public	-	-
getTotalYesVotes	Public	-	-
getVotingStart	Public	-	-
getResults	External	-	-
allow	External	Can modify state	onlyOwner
submit	External	Can modify state	onlyAllowedSubmitters onlyNotExistingLoans
retract	External	Can modify state	onlyPendingLoans onlyCreator
vote	Internal	Can modify state	-
yes	External	Can modify state	onlyPendingLoans
no	External	Can modify state	onlyPendingLoans
withdraw	External	Can modify state	onlyNotRunningLoans
bounty	Internal	-	-
toTrustToken	Internal	-	-
claim	Public	Can modify state	onlyFundedLoans calculateTotalReward
wasPredictionCorrect	Internal	-	-
status	Public	-	-

4.2 Code Audit

4.2.1 Medium-risk vulnerabilities

4.2.1.1 DoS Issue

Borrowers can continuously add loan tokens with the logic `_loan.push(loanToken)` in fund() function，it should still avoid excessive recursion, as every internal function call uses up at least one stack slot and there are only 1024 slots available. when _loans.length is too large there will be DoS.

- TrueLender.sol Line:117-183

```
for (uint256 index = 0; index < _loans.length; index++) {
            if (_loans[index] == loanToken) {
                _loans[index] = _loans[_loans.length - 1];
                _loans.pop();
                break;
            }
        }
```

- TrueLender.sol Line:591-597

```
    function value() external override view returns (uint256) {
        uint256 totalValue;
        for (uint256 index = 0; index < _loans.length; index++) {
            totalValue = totalValue.add(_loans[index].value(_loans[index].balanceOf(address(this))));
        }
        return totalValue;
    }
```

- TrueLender.sol Line:638-639

```
function distribute(
        address recipient,
        uint256 numerator,
        uint256 denominator
    ) external override onlyPool {
        for (uint256 index = 0; index < _loans.length; index++) {
            _loans[index].transfer(recipient, numerator.mul(_loans[index].balanceOf(address(this))).div(denominator));
```

```
    }
}
```

suggestion for fix:

It is suggested to avoid excessive recursion, you can use the Mapping data structure to record LoanToken, and then find it through mapping, you need to add a batch processing mechanism, when DoS occurs, you can maintain normal operation through batch processing.

4.2.1.2 Race Conditions issue

There has a Race Conditions issue. When the status is Status.Awaiting, any user can call the fund() function to become a lender, where the attacker can preemptively call the fund() function by increasing the gas price.

When Bob creates LoanToken and wants to become lender by calling the fund() function, but Alice calls the fund() function first to become the lender by increasing the gas price and no need to spend ETH to create a LoanToken contract.

- LoanToken.sol Line: 230-238

```
function fund() external override onlyAwaiting {
    status = Status.Funded;
    start = block.timestamp;
    lender = msg.sender;
    _mint(msg.sender, debt);
    require(currencyToken.transferFrom(msg.sender, address(this), receivedAmount()));
    emit Funded(msg.sender);
}
```

suggestion for fix:

It is suggested to add permission check.

4.2.2 Low-risk vulnerabilities

4.2.2.1 Missing permission check

There is an permission issue, here attacker can arbitrarily transfer the token of the authorized user when "status >= Status.Withdrawn".

When Bob approve to LoanToken with 1000 amount, but only wants to transfer 500 amount to LoanToken contract now, at this time, the attacker can maliciously transfer the remaining 500 amount to LoanToken contract.

- LoanToken.sol Line: 296

```
function repay(address _sender, uint256 _amount) external override onlyAfterWithdraw {
    require(currencyToken.transferFrom(_sender, address(this), _amount));
}
```

suggestion for fix:

It is suggested to add permission check.

4.2.3 Enhancement Suggestions

4.2.3.1 Missing emit

It is suggested to add the emit keyword ahead of FarmChanged event to optimize coding standards.

- distributors/LinearTrueDistributor.sol Line:71

```
function setFarm(address newFarm) external onlyOwner {
    farm = newFarm;
    FarmChanged(newFarm);
    }
```

suggestion for fix:

```
function setFarm(address newFarm) external onlyOwner {
        farm = newFarm;
        emit FarmChanged(newFarm);
    }
```

4.2.3.2 No parameter variable name

It is suggested to add function parameter names to optimize the coding standards.

- distributors/LinearTrueDistributor.sol Line:77

```
function distribute(address) public override
```

```
interface ITrueDistributor {
    function trustToken() external view returns (IERC20);
    function distribute(address farm) external;
    function empty() external;
}
```

suggestion for fix:

```
function distribute(address farm) public override
```

4.2.3.3 Redundant code

The hook function has no specific implementation it is redundant code.

- upgradeability/UpgradeableERC20.sol Line:323-328

```
function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
```

suggestion for fix:

It is suggested to remove redundant code to optimize Gas.

4.2.3.4 Enhanced suggestions for transfering ownership

According to the current logic, when the ownership is transferred, directly use the transferOwnership function to transfer the ownership to newOwner.

- upgradeability/UpgradeableOwnable.sol Line: 65-69

```
function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

suggestion for fix:

It is suggested to use the acceptOwnership function, The ownership can be transferred only after the newOwner is accepted.

```
  function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _newOwner = newOwner;
    }
}
```

```
modifier onlyNewOwner() {
        require(_newOwner == msg.sender, "Ownable: caller is not the newOwner");
        _;
    }
```

```
function acceptOwnership() public virtual onlyNewOwner {
        emit OwnershipTransferred(_owner, _newOwner);
        _owner = _newOwner;
       _newOwner = address(0);
    }
}
```

4.2.3.5 External data missing check

Relying on curve's prices and calculated data, missing checks and limiting the maximum allowable fluctuation range.When external data is maliciously manipulated, it may affect the normal operation of TrueFi.

- TrueFiPool.sol Line:150

```
function poolValue() public view returns (uint256) {
        // prettier-ignore
        return
            currencyBalance()
            .add(_lender.value())
            .add(
                yTokenBalance()
                .mul(_curvePool.curve().get_virtual_price())
                .div(1 ether));
    }
```

- TrueFiPool.sol Line:365-371

```
function calcTokenAmount(uint256 currencyAmount) public view returns (uint256) {
        // prettier-ignore
        uint256 yTokenAmount = currencyAmount.mul(1e18).div(
_curvePool.coins(TUSD_INDEX).getPricePerFullShare());
        uint256[N_TOKENS] memory yAmounts = [0, 0, 0, yTokenAmount];
        return _curvePool.curve().calc_token_amount(yAmounts, true);
    }
```

- TrueFiPool.sol Line:378-380

```
function calcWithdrawOneCoin(uint256 yAmount) public view returns (uint256) {
        return _curvePool.calc_withdraw_one_coin(yAmount, TUSD_INDEX);
    }
```

suggestion for fix:

It is suggested to check external data and limit the fluctuation range.

5. Audit Result

5.1 Conclusion

Audit Result : Medium-Risk

Audit Number : 0X002011200002

Audit Date : Nov. 20, 2020

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, 2 medium-risk vulnerabilities, 1 low-risk vulnerabilities, 5 enhancement suggestion were found during the audit.

6. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.
For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist