PROTOCOL

SMART CONTRACTS AUDIT

zokyo

September 12th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

TECHNICAL SUMMARY

This document outlines the overall security of the TProtocol smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the TProtocol smart contracts codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the TProtocol team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the TProtocol repository: https://github.com/TProtocol/USTP/

Last commit - `0aba658`

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- InterestRateModel.sol
- iUSTP.sol
- LiquidatePool.sol
- migrator.sol
- rUSTP.sol
- rUSTPool.sol
- USTP.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TProtocol smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contracts by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

No critical issues were discovered during the audit, but we did identify several of medium and low severity, as well as a couple of informational concerns. These are elaborated upon extensively in the "Complete Analysis" segment.

TP V2 Mainnet Address (Update on Sept 8, 12:48 ET)
liquidatePool: "0×2B11279F01998587cB03e0fc18797aAA8c4300f6",
rustPool: "0×38a1753AEd353e58c64a55a3f3c750E919915537",
interestRateModel: "0xA8D1355868Adb65543c90D25d79FC3d72A30906d",
ustp: "0xed4d84225273c867d269F967CC696e0877068f8a",
iustp: "0×36df9B0F5e50b6F1341e8D90b222dAa0B5dc385b",
migrator: "0xd85E55040B1B4737e94004B0CE8592BDa03293f4"

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the TProtocol team and the TProtocol team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

- **Critical**

  The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

- **High**

  The issue affects the ability of the contract to compile or operate in a significant way.

- **Medium**

  The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

- **Low**

  The issue has minimal impact on the contract’s ability to operate.

- **Informational**

  The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

System Overview

The main use of the TProtocol contracts audited by the Zokyo team is to serve as a platform for RWA assets such as treasury bills. As treasury bills are considered to be the risk-free rate in the world. TProtocol has built a system that tries to capitalize on this by bringing to on-chain. During the manual and testing stages of the contracts audit, multiple security issues were found. All those can be found in the following section. Besides these findings, there are also remarks that have to be made about the overall security of the contracts submitted for audit.

FINDINGS SUMMARY

#	Title	Risk	Status
1	Lack of Input Validation for Liquidation Threshold	Medium	Acknowledged
2	APR Unintentionally Set to High Value	Medium	Resolved
3	Missing Checks For Whether Arbitrum Sequencer Is Active	Medium	Acknowledged
4	Funds Can Be Permanently Lost if PK leak	Medium	Acknowledged
5	Centralization Risk	Medium	Acknowledged
6	Deprecated Chainlink Usage	Medium	Resolved
7	Reserves MightBe Calculated As 0 Due To Rounding	Medium	Acknowledged
8	Mixed up shares/rUSTP conversion	Medium	Resolved
9	Incorrect Share Calculation	Medium	Resolved
10	Unsafe ERC-20 Transfer	Low	Acknowledged
11	Missing Pause Condition From transferShares Function	Low	Acknowledged
12	convertToUSDC Would Be Calculated Incorrectly If USTP Amount Is Below One	Low	Resolved
13	Input Value of j is Not Validated Combined With Unsafe Casting	Low	Acknowledged
14	`withdrawUSDC` Can Cause Unexpected Losses	Low	Resolved
15	Hard Coded Stablecoin Price	Low	Acknowledged
16	“repayAll” Executed With Zero Shares	Informational	Resolved
17	Wasted computation on a paused contract	Informational	Resolved
18	Duplicate Logic	Informational	Acknowledged
19	Missing Emit Events	Informational	Resolved

#	Title	Risk	Status
20	Remaining To do and Missing Functionality	Informational	Acknowledged
21	Missing Documentation	Informational	Acknowledged
22	Floating Pragma	Informational	Resolved
23	Custom Errors	Informational	Acknowledged
24	Typos	Informational	Resolved
25	transferShares triggered for same address sender/ recipient	Informational	Acknowledged

MEDIUM | ACKNOWLEDGED

Lack of Input Validation for Liquidation Threshol

rUSTPool.sol - Functions liquidateBorrow and flashLiquidateBorrow can be called by any wallet to liquidate a user's borrow position with any amount, even though the debt position of the user is healthy. Because anyone can liquidate a wallet, a bot can be set up to listen to borrowUSDC function calls in the mem-pool. When a user calls borrowUSDC the bot instantly liquidates them, griefing the user by not being able to use the protocol and losing yield.

Recommendation:
It is advised to enable the liquidation within a certain specified threshold that is related to depositedSharesSTBT and borrowedAmount. A similar formula to the one applied in _requireIsSafeCollateralRate needs to be implemented for that scenario.

MEDIUM | RESOLVED

APR Unintentionally Set to High Value

In InterestRateModel.sol - It is noted from setAPR() that the value of APR should be less than 8e5. Despite that the initial value of APR is being set to 42e5. It is pointed out in the comments that it is meant to be 4.2% which is 4.2e5.

```
// Assuming the maximum is 4.2%
uint256 private APR = 42 * 1e5;

function setAPR(uint256 newAPR) external onlyRole(DEFAULT_ADMIN_ROLE) {
 require(newAPR <= 8 * 1e5, "apr should be less then 8%");
 ...
}
```

Recommendation:
Initial value of APR needs to be changed or adjust the comments to be consistent with the code.

MEDIUM | ACKNOWLEDGED

Missing Checks For Whether Arbitrum Sequencer Is Active

The protocol intends to deploy to L2 chains as well.
Chainlink recommends that users using price oracles, check whether the Arbitrum sequencer is active
https://docs.chain.link/data-feeds#l2-sequencer-uptime-feeds
If the sequencer goes down, oracles may have stale prices, since L2-submitted transactions (i.e. by the aggregating oracles) will not be processed.

Recommendation:
Use sequencer oracle inside the LiquidatePool.sol contract , else it might give stale prices for USDC and affect the liquidation process where we demand a certain peg from the returned amount.

MEDIUM | ACKNOWLEDGED

Funds Can Be Permanently Lost if PK leak

In the contract, `LiquidatePool.sol,` an issue can arise if the private key is compromised. This can lead to unintended consequences if the processing time is changed and someone is trying to call `finalizeLiquidationById()`

Recommendation:
Use a multi-sig wallet to combat this issue

Centralization Risk

The wallet behind the `POOL_MANAGER_ROLE` and `DEFAULT_ADMIN_ROLE` wallet has excessive control over the contracts in the audit. The centralized `POOL_MANAGER_ROLE` and `DEFAULT_ADMIN_ROLE` wallets have control over the following functions:

- `pause()` - toggle the usability of the protocol.
- `unpause()` - toggle the usability of the protocol.
- `initLiquidatePool()` - set a liquidate pool that they chose.
- `initMigrator()` - change the migrator.
- `revokeMigrator()` - revoke the migrator from being able to exchange old tokens.
- `claimReservesFee()` - claim protocol fees.
- `setReserveFactor()` - set the reserve factor to change the capability of loans
- `setInterestRateModel()` - increase or decrease the interest rate for the protocol.
- `setSafeCollateralRate()` - change the collateralization for the protocol
- `acceptFlashLiquidateProvider()` - Allow a user to be a flash loan provided for the Curve pool.
- `RecoverERC20()` - Withdraw any non-rUSTP from the protocol.
- `claimUSTP()` - claim a specific amount of rUSTP that needs to be recovered.
- `setBorrower()` - set the borrower for the migration contract.
- `setAPR()` - set a new APR for the protocol.
- `setProcessPeriod()` - change the withdrawal time from liquidating USDC.
- `setLiquidateFeeRate()` - set the fee for liquidation.
- `setFeeCollector()` - set a new wallet to collect protocol fees.
- `setReedemMXPFeeRate() - change the fee that Matrix Portal receives from liquidations.
- `setReedemPool()` - set a new Matrix Portal redemption pool.
- `setCurvePool()` - change the curve pool.
- `setReedemThreshold()` - change the value for redemption of Matrix Pool.
- `setPegPrice()` - set the price for USDC

Recommendation:

We recommend using a timelock for sensitive functions such as `RecoverERC20()` and using a multi-sig for all centralized functions to ensure that the protocol cannot incur large losses from a leaked private key.

MEDIUM | RESOLVED

Deprecated Chainlink Usage

In the contract, `LiquidatePool.sol`, the function latestAnswer is a deprecated function call. If the call returns stale data, then it could return zero as the result and cause unintended consequences. It is recommended to change to lastestRoundData to get the price instead. ChainLink does not recommend using latestAnswer in their documentation.

Recommendation:
We recommend using lastestRoundData instead of latestAnswer.

MEDIUM | ACKNOWLEDGED

Reserves MightBe Calculated As 0 Due To Rounding

Inside rUSTPool.sol inside modifier realizeInterest at L95 , we calculate reserves depending upon the total interest accrued. In case where reserveFactor is close to FEE_COEFFICIENT and totalInterest accrued is high enough , the reserves value at L98 would be calculated as 0 due to rounding.
Example → reserveFactor = 1e3 , in this case, if totalInterest is less than 1e5 the reserves would be calculated as 0 and be not accounted for total unclaimed reserves and rUSTP supply.

Recommendation:
Have proper normalization done for the amount of the reserve or ensure that reserves to not be 0.

MEDIUM | RESOLVED

Mixed up shares/rUSTP conversion

rUSTP.sol - Function _burnShares() in lines: 272, 276 _sharesAmount is supposed to be converted to rUSTP but the method getSharesByrUSTPAmount is used which is expected from its formula to convert rUSTP amount to shares amount.

```
272     uint256 preRebaseTokenAmount = getSharesByrUSTPAmount(_sharesAmount);
273     newTotalShares = _getTotalShares().sub(_sharesAmount);
274     totalShares = newTotalShares;
275     shares[_account] = accountShares.sub(_sharesAmount);
276     uint256 postRebaseTokenAmount = getSharesByrUSTPAmount(_sharesAmount);
```

Recommendation:

Function getrUSTPAmountByShares() needs to be used instead.

MEDIUM | RESOLVED

Incorrect Share Calculation

During testing the client spotted an issue with the share calculation.The issue can occur if the protocol has less than 100% utilization rate for borrowing USDC. When a user borrows USDC, the amount of shares is incorrectly calculated. When the user pays back the USDC, they can pay a lower amount than intended leaving a shortfall in the contract as rUSTP is rebased at a higher value. To fix this, the borrow needs to be divided by the total amount borrowed instead of by the total supply of rUSTP.

The client correctly fixed this by creating new functions to correctly check the share values based on the total borrowed shares instead of the total supply of rUSTP. The fix was corrected in commit: 0aba6588a763ab82c0798f0fe6c32a1ff9a45cd8

Unsafe ERC-20 Transfe

Contracts include several occurrences of unsafe transfer of ERC20 tokens. This might lead to undesirable results if the transfer is not successful while the transaction is.

LiquidatePool.sol - Function finalizeLiquidationById()

usdc.transfer(msg.sender, redeemAmountAfterFee);
usdc.transfer(feeCollector, protocolFee);

In rUSTPool.sol -
In supplyUSDC() the transfer of usdc from msg.sender is not being ensured to fulfill by a safe transfer.
As well in _supplySTBTFor()
stbt.transferFrom(msg.sender, address(this), _amount);

As well in withdrawSTBT
stbt.transfer(msg.sender, _amount);

In withdrawUSDC
usdc.transfer(msg.sender, _amount);

In borrowUSDC
usdc.transfer(msg.sender, _amount);

In repayUSDC
usdc.transferFrom(msg.sender, address(this), _amount);

In repayAll()
usdc.transferFrom(msg.sender, address(this), convertToUSDC);

In liquidateBorrow()/flashLiquidateBorrow()
stbt.transfer(address(liquidatePool), repayAmount);

In migrate()
stbt.transferFrom(_borrower, address(this), _amount);

Recommendation:
Utilize SafeERC20 for token transfers.

Fix: As of commit b092c1c issue is fixed in LiquidatePool.sol and only one occurrence in rUSTPool.sol while the all the other transfers remain unaddressed.

LOW | ACKNOWLEDGED

Missing Pause Condition From transferShares Function

Inside rUSTP.sol , at L159 for the function transferShares it is mentioned that contract must not be paused , but there is no pause functionality implemented , either it is overlooked or the comment is deprecated.

LOW | RESOLVE

convertToUSDC Would Be Calculated Incorrectly If USTP Amount Is Below One

Inside rUSTPool.sol at L357 we calculate convertToUSDC by dividing by 1e12 . It's not necessary that repayrUSTP is at least 1 USTP , so if repayrUSTP is less than 1e12 the USDC amount calculated would be incorrect.

Recommendation:
Proper normalization should be done for the cases where repayrUSTP is less than 1 or ensure that repayrUSTP is at least 1.

LOW | ACKNOWLEDGED

Input Value of j is Not Validated Combined With Unsafe Casting

In LiquidatePool.sol - Function flashLiquidateSTBTByCurve() does not validate the value of j to be within the accepted range. It is shown here that uint256(int256(j-1)) should be within the bounds of coins array length.

```
IERC20 targetToken = IERC20(coins[uint256(int256(j - 1))]);
```

Recommendation:
Require j to be within the expected valid range to avoid unexpected results.

LOW | RESOLVE

withdrawUSDC` Can Cause Unexpected Losse

As rUSTP is an interest bearing token, the price should positively reflect over time. As the value goes up, it will be worth more than $1. If a user calls `withdrawUSDC()`, they can have leftover dust and think they forfeited their money. As this can be avoided by adding a function that withdraws all.

Recommendation:
We recommend changing the function name to `EmergencyUSDCWithdraw()` as depending on the deposit size of the user. The loss of interest could be detrimental to the user.

Fix: in commit: `b092c1c80c1c77701a1b1030a72638ba8492f91c`

LOW | ACKNOWLEDGED

Hard Coded Stablecoin Pric

Because the price of STBT is hardcoded to 1, this can cause issues if STBT starts to de-peg and can cause many unintended consequences

Recommendation:
It is always recommended to use an Oracle instead of hardcoding the price.

INFORMATIONAL | RESOLVED

repayAll Executed With Zero Shares

rUSTPool.sol - Function repayAll() triggers the user to repay the borrowedShares[msg.sender]. Even if the borrowedShares[msg.sender] is zero. No require statement to revert early with an error message that describes the negative result of such a transaction.

Recommendation:
add require statement that ensures msg.sender does have non-zero borrowedShares

INFORMATIONAL | RESOLVED

Wasted computation on a paused contract

rUSTPool.sol - The modifier realizeInterest precedes whenNotPaused in
function supplyUSDC(uint256 _amount) external realizeInterest whenNotPaused
Modifier realizeInterest is costly in terms of computation and all the computation is wasted if contract is already paused so its better to check if contract paused or not before going through the body of realizeInterest.

Recommendation:
switch modifiers order.

INFORMATIONAL | ACKNOWLEDGED

Duplicate Logi

In iUSTP.sol - Functions unwrapAll() and unwrap() contains duplicate logic. So in USTP.sol - Functions unwarpAll() and withdraw() And in rUSTPool.sol - Functions flashLiquidateBorrow() and liquidateBorrow().

Recommendation:
Implement a private function for the common logic (i.e. _unwrapProcedure(uint256 _shares)).

Fix: This informational note is addressed in the most significant occurrence in rUSTPool.sol but ignored elsewhere.

Missing Emit Events

LiquidatePool.sol - No events emitted in functions that trigger a change of significant parameters by admin. As users can use this to monitor centralized changes to the protocol. Emitted events are always welcome and help bring transparency to the protocol.

```
function setProcessPeriod(uint256 _processPeriod) external onlyAdmin
function setFeeCollector(address _feeCollector) external onlyAdmin
function setLiquidateFeeRate(uint256 _liquidateFeeRate) external onlyAdmin
function setRedeemMXPFeeRate(uint256 _liquidateMXPFeeRate) external onlyAdmin
function setRedeemPool(address _redeemPool) external onlyAdmin
function setCurvePool(address _curvePool) external onlyAdmin
function setRedeemThreshold(uint256 amount) external onlyAdmin
function setPegPrice(int256 _targetPrice) external onlyAdmin
```

migrator.sol -

```
function setBorrower(address _borrower) external onlyAdmin
```

Recommendation:
Emit an event in the body of the setter functions.

INFORMATIONAL | ACKNOWLEDGED

Remaining Todo and Missing Functionalit

In the contract, `rUSTPool.sol`, there are two TODO's left on line 387 and 429. It is recommended that these be solved and implemented or removed with proper reasoning in the removal.

Recommendation:
The contract is also missing a potential `whenNotPaused()` modifier on `transferShares()`. The comment lines should be updated if this is not intended or the function should have the added modifier.

Fix: Partially Resolved

INFORMATIONAL | ACKNOWLEDGED

Missing Documentation

The contracts are highly complex in how they interact with each other. The documentation should have proper details for outside reviewers and developers to understand how the protocol works and interacts with STBT properly.

Recommendation:
It is recommended that more documentation is written to accurately describe what the intended behavior of a smart contract is and its intended functionality.

INFORMATIONAL | RESOLVED

Floating Pragma

Throughout the codebase, the contracts that are unlocked at version ^0.8.18, and it should always be deployed with the same compiler version. By locking the pragma to a specific version, contracts are not accidentally getting deployed by using an outdated version that can introduce unintended consequences.

Recommendation:
Lock the compiler version to a specific one. Known bugs are featured here

INFORMATIONAL | ACKNOWLEDGED

Custom Errors

Throughout the codebase, `require` statements are used instead of custom errors. Custom errors are available from solidity version ^0.8.4. Custom errors can roughly save 50 gas per call. This gas is saved because of not having to allocate the string and store the string through the revert. Throughout the contracts in the codebase, `require` statements are used instead of custom errors. Custom errors save gas on deployment as well.

Recommendation:
Convert `require` statements to custom errors to save gas on each function call and deployment

Typos

Throughout the codebase, there are many typos in comments, variable names, and function names. These typos should be fixed for the production environment to ensure all outside reviewers do not misinterpret the intended behavior of the protocol.
The following typos occur:

- Function `unwarpAll()` → should be spelled `unWrapAll()`
  - Exists in contract `USTP.sol` and `IUSTP.sol`.
- `warp()` should be changed to `wrap()`.
- * @dev Allows to recovery nUSTP incorrect grammar should be changed to * @dev Allows recovery of nUSTP.
- * @dev Allows to recovery any ERC20 token should be changed to * @dev Allows recovery of any ERC20 token except `rUSTP`.
- * @dev unwarp iUSTP to rUSTP should be changed to * @dev unwrap iUSTP to rUSTP.
- * @dev warp rUSTP to iUSTP should be changed to * @dev wrap rUSTP to iUSTP.
- `lqiuidateShares` in nUSTP is spelled incorrectly and should be changed to `liquidateShares`.
- `lqiuidateAmount` should be changed to `liquidateAmount`.
- // At migrate. we don't check healthy should be changed to be cleare

Recommendation:
We recommend fixing the typos.

transferShares triggered for same address sender/recipient

In rUSTP.sol - Function `transferShares()` is triggered even when `recipient` is the sender of shares. This takes place since function `_transferShares(address _sender, address _recipient, uint256 _sharesAmount)` also does not validate the `_sender` and `_recipient` not same address.

Recommendation:
Require statement to ensure addresses are not equal.

	InterestRateModel.sol iUSTP.sol LiquidatePool.sol migrator.sol rUSTP.sol rUSTPool.sol USTP.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the TProtocol team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the TProtocol team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo