TRZNFinance

SMART CONTRACTS REVIEW

zokyo

January 12th 2024 | v. 1.0

Security Audit Score

PASS

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

SCORE
94

ZOKYO AUDIT SCORING TRZN

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 1 Critical issue: 1 resolved = 0 points deducted
- 4 High issues: 4 resolved = 0 points deducted
- 10 Medium issues: 9 resolved and 1 acknowledged = - 3 points deducted
- 15 Low issues: = 12 resolved and 3 acknowledged = - 3 points deducted
- 16 Informational issues: 14 resolved and 2 acknowledged = 0 points deducted

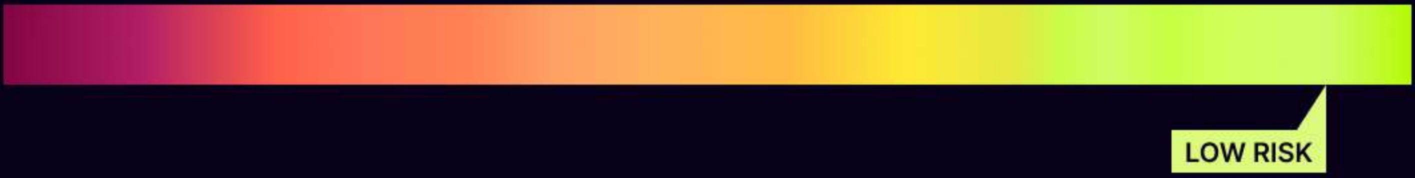
Thus, 100- 3 - 3 = 94

TECHNICAL SUMMARY

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

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

Contract Status



There was 1 critical issue found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contracts that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the TRZN 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 TRZN repository:
Repo: https://github.com/digitaldruid218/ZeUSD_contract_v01/tree

Last commit - e65cd059b6fd8a56408d3ff04c4092adecb19227

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

- ./Contract_V002/interface/IStable.sol
- ./Contract_V002/interface/IRiskManagerEscrow.sol
- ./Contract_V002/interface/IEscrowToken.sol
- ./Contract_V002/interface/IUniswapV3Pool.sol
- ./Contract_V002/interface/IERC20PriceOracle.sol
- ./Contract_V002/interface/IVaultManager.sol
- ./Contract_V002/interface/IUniswapV2Pair.sol
- ./Contract_V002/library/Type.sol
- ./Contract_V002/library/RiskManagerEscrowLibrary.sol
- ./Contract_V002/library/OracleLibrary.sol
- ./Contract_V002/library/Utils.sol
- ./Contract_V002/MultiSig_V2.sol
- ./Contract_V002/RiskManagerEscrow_V2.sol
- ./Contract_V002/ERC20PriceOracle_V2.sol
- ./Contract_V002/TokenStableV6_V2.sol
- ./Contract_V002/SwapOne2One_V2.sol
- ./Contract_V002/VaultETH_V2.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 resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TRZN 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

The Zokyo team conducted an in-depth assessment of TRZN's codebases. During the evaluation, a critical issue was discovered, and findings of high, medium, low, and informational significance were identified. The TRZN team promptly addressed and commented on these issues. Comprehensive descriptions of these findings are available in the "Complete Analysis" section.

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 TRZN team and the TRZN 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

FINDINGS SUMMARY

#	Title	Risk	Status
1	Missing access control in burn	Critical	Resolved
2	Missing access control check	High	Resolved
3	Reentrancy attack is possible on SellETH function	High	Resolved
4	Function RM_UpdateReward in VaultETH_V2 has no access control	High	Resolved
5	Risk Managers Are Not Paid Out In Request_BuyETH	High	Resolved
6	Use SafeTransfer Instead Of Transfer	Medium	Resolved
7	Use SafeTransfer Instead Of Transfer	Medium	Resolved
8	Wrong avgDepositPrice calculated	Medium	Resolved
9	Centralization risk	Medium	Acknowledged
10	Require check in MultiSig_V2 can be bypassed	Medium	Resolved
11	Uncalled base constructor in MultiSig	Medium	Resolved
12	Missing Fallback Function to reject accidental Ether transfers to contract StableSwap_V2	Medium	Resolved
13	Missing reentrancy guard in EmergencyWithdraw / WithdrawEscrow	Medium	Resolved
14	Possible Integer Overflow/Underflow in quicksort function in utils library	Medium	Resolved
15	No Price Staleness Check	Medium	Resolved
16	Missing checks for failures in getChainlinkPrice Function in ERC20PriceOracle_V2	Low	Acknowledged

#	Title	Risk	Status
17	2-step ownership transfer	Low	Resolved
18	calcFee Would Return Incorrect Value For A Fee-On-Transfer targetToken	Low	Acknowledged
19	No check for address(0)	Low	Resolved
20	Use .call() for transferring ETH	Low	Resolved
21	Missing mechanism to track Votes	Low	Resolved
22	Typing error in the RemoveContract_toStableToken() function	Low	Resolved
23	Transfer ownership to zero address in RiskManagerEscrow_V2	Low	Resolved
24	Missing zero value check for gap in RiskManagerEscrow_V2	Low	Resolved
25	Unsafe Downcasting in TokenStableV6_V2	Low	Resolved
26	MIssing sanity value checks in the constructor of SwapOne2One	Low	Resolved
27	Add openzeppelin's safeMath Library in StableSwap_V2 and VaultETH_V2	Low	Acknowledged
28	Transfers To The Risk Managers Might Revert	Low	Resolved
29	Use Of slot0 To Get sqrtPriceX96 Can Lead To Price Manipulation	Low	Resolved
30	Withdraw Function Might Revert If Not Sufficient Balance Of Tokens	Low	Resolved
31	Manipulate112 Should Manipulate To uint112 Instead Of 128	Informational	Resolved
32	Missing checks for parameters for setParams Function in ERC20PriceOracle_V2	Informational	Resolved

#	Title	Risk	Status
33	2-step ownership Missing check for array length greater than 0 getERC20Price_High and getERC20Price_Low Functions	Informational	Resolved
34	CEI not followed in WithdrawEscrow()	Informational	Resolved
35	Missing checks in CheckRiskManagerStatus function	Informational	Acknowledged
36	Unused params	Informational	Resolved
37	Repetitive condition check	Informational	Resolved
38	Events parameters not indexed	Informational	Resolved
39	No validation when setting parameters	Informational	Resolved
40	CEI not followed in WithdrawEscrow()	Informational	Resolved
41	Floating pragma	Informational	Resolved
42	Typos and Incorrect comment	Informational	Resolved
43	The SwapOne2One_V2 pool assumes token value to be constant	Informational	Acknowledged
44	RiskManagerEscrow Cannot Receive Ether	Informational	Resolved
45	Directly Calling The calcFee Function Will Result In Lost Ether	Informational	Resolved
46	Unnecessary Assignment	Informational	Resolved

CRITICAL-1 | RESOLVED

Missing access control in burn

In the TokenStableV6 contract, there is a missing access control in burn(). This function can be used by an attacker to burn tokens from any address.

Recommendation:

It is advised to either allow only the msg.sender to burn his own tokens, or add an appropriate access control modifier for the same.

HIGH-1 | RESOLVED

Missing access control check

In Contract VaultETH_V2, the method RM_UpdateReward(...) should only be callable by an account that has the RISK_MANAGER role as per the comment similar to the method RM_UpdateDeposit(...).
Since this method has no access control check, malicious users can call this method which can cause DoS for other users to mint ZeUSD tokens as this method internally calls the `SellETH()` method which can mint max amount of ZeUSD as the `SellETH()` method allows this contract to mint as many tokens as possible with no upper limit.

Recommendation:

Add the following check in the method RM_UpdateReward().

```
        assert(hasRole(RISK_MANAGER, msg.sender));
```

HIGH-2 | RESOLVED

Reentrancy attack is possible on SellETH function

The SellETH function sends Ether to risk managers, if these risk managers are contracts that have malicious code, there is a possibility of reentrancy attack. Even though this contract inherits ReentrancyGaurd, the guard will only apply to functions that have nonreentrant modifier.

Recommendation:

To mitigate this risk, it's generally a good practice to mark the function with nonreentrant modifiers from openzeppelin.

Also place Ether transfer operation at the end of function. By doing this, the transfer operation occurs after all state changes are completed, reducing the risk of reentrancy attacks.

HIGH-3 | RESOLVED

Function RM_UpdateReward in VaultETH_V2 has no access control

Function RM_UpdateReward has no access control and therefore can be called by any user even though the function seems to have to be called by risk managers.

Recommendation:

Mitigate these issues by performing checks to ensure that function is not called by unexpected addresses

HIGH-4 | RESOLVED

Risk Managers Are Not Paid Out In Request_BuyETH

A user can request to buy ETH from the protocol in exchange with the stable tokens using the function Request_BuyETH in VaultETH_V2 . A portion of the ETH that should be sent to the user is reserved for the risk managers which is calculated at L380.
After calculation of these amount the risk managers are not paid , just the amount to be transferred is calculated and stored into a local storage uint array.

Recommendation:

Add the transfers for the risk managers.

MEDIUM-1 | RESOLVED

Use SafeTransfer Instead Of Transfer

In the swapBforA function (SwapOnetoOne_V2.sol) there is a ERC20 transfer taking place at L90 . The return value of the transfer is not checked so it is possible that the transfer fails silently (returning a false) and the rest of the function executes normally . In that case token balances and fees would be updated without any transfer taking place.

Recommendation:

Use safeTransfer or check the return value of the transfer

MEDIUM-2 | RESOLVED

Wrong avgDepositPrice calculated

In Contract VaultETH_V2, the method Request_BuyETH(...) allows users to buy ETH using ZeUSD tokens.

In the method, the epoch's average deposit price is calculated as follows:

```
tempEpochInfo.avgDepositPrice =
        (tempEpochInfo.rmDeposit *
           tempEpochInfo.avgDepositPrice +
           sendAmt *
           ethPrice) /
        (tempEpochInfo.rmDeposit + sendAmt);

     *// Update the total deposit Ether amount for the epoch*
     tempEpochInfo.rmDeposit += ethAmt;
```

tempEpochInfo.avgDepositPrice is the weighted average ETH price when the user calls the "Request_BuyETH" function, but this formula is using ZeUSD tokens amount for calculating the same which is incorrect.

Recommendation:

Update the above formula as follows:
tempEpochInfo.avgDepositPrice = (tempEpochInfo.rmDeposit * tempEpochInfo.avgDepositPrice + ethAmt * ethPrice) / (tempEpochInfo.rmDeposit + ethAmt);

MEDIUM-3 | ACKNOWLEDGED

Centralization risk

In Contract VaultETH_V2, there are several methods using the modifier `onlyRole(DEFAULT_ADMIN_ROLE)` which can be used to configure important parameters for the protocol such as oracles, WETH, etc. But in the constructor, DEFAULT_ADMIN_ROLE is being set to msg.sender which is an EOA.

There is also an emergencyWithdraw method which allows DEFAULT_ADMIN_ROLE to withdraw all the deposited ETH in the vault.

This risks the whole protocol being centralized and controlled by a single EOA.

Recommendation:
It is advised to decentralize the usage of these functions by using a multisig wallet with at least 2/3 or a 3/5 configuration of trusted users. Alternatively, a secure governance mechanism can be utilized for the same.

Client comment: We originally planned to deploy multi-sig and update it in the future by replacing the admin address with multi-sig. If you would recommend ways to improve contract code stability, we would appreciate it so we may consider how to improve.

MEDIUM-4 | RESOLVED

Require check in MultiSig_V2 can be bypassed

In the MultiSig contract, the require check statement of _numConfirmationsRequired <= _whiteWallet.length could be bypassed if DeregisterWhiteWallet() is used. For example, if the wallet _whiteWallet.length is 3 and the _numConfirmationsRequired is also 3, it is possible that DeregisterWhiteWallet() would be called in future.

After the call, the _numConfirmationsRequired would become 3 whereas the _whiteWallet.length would be 2, which would be a violation of the statement in constructor which tries to ensure that _numConfirmationsRequired <= _whiteWallet.length.

```
 require(
   _numConfirmationsRequired > 0 &&
      _numConfirmationsRequired <= _whiteWallet.length,
   "invalid number of required confirmations"
);
```

Recommendation:

It is advised to add an internal function that decreases the _numConfirmationsRequired accordingly in the DeregisterWhiteWallet() function.

MEDIUM-5 | RESOLVED

Uncalled base constructor in MultiSig

The base constructor of the Ownable contract is not called in the MultiSig contract. This means that the initialOwner variable of the Ownable contract is not assigned.

Recommendation:

It is advised to initialize the constructor of the Ownable contract with a call to Ownable(initialOwner) in the constructor of the MutiSig_V2 contract.
For example: constructor(address[] memory _whiteWallet, uint _numConfirmationsRequired, address _stableToken, address initialOwner) Ownable (initialOwner){}

MEDIUM-6 | RESOLVED

Missing Fallback Function to reject accidental Ether transfers to contract StableSwap_V2

Recommendation:

Consider implementing a non-payable fallback function with appropriate logging to handle any unexpected Ether transfers to this contract in order to prevent Ether from being stuck in the contract.

MEDIUM-7 | RESOLVED

Missing reentrancy guard in EmergencyWithdraw / WithdrawEscrow

By adding the nonreentrant modifier, you can help ensure that the function is not vulnerable to reentrancy attacks, especially when interacting with external contracts.

MEDIUM-8 | RESOLVED

Possible Integer Overflow/Underflow in quicksort function in utils library

The use of uint(left + (right - left) / 2) might result in an overflow if the right is a very large negative number. Consider validating the array indices to avoid potential overflow issues.

MEDIUM-9 | RESOLVED

No Price Staleness Check

The function latestRoundData() has been used inside ERC20PriceOracle_V2 (getChainlinkPrice) to fetch the price of an asset , but there are no price staleness checks.

Recommendation:

Introduce price staleness checks as follows →

```
(uint80 roundID, int256 answer, , uint256 timestamp, uint80 answeredInRound) =
tokenInfos[token].chainlinkOracle.latestRoundData();
require(answeredInRound >= roundID, "Stale price");
require(timestamp != 0,"Round not complete");
```

LOW-1 | ACKNOWLEDGED

Missing checks for failures in getChainlinkPrice Function in ERC20PriceOracle_V2

The getChainlinkPrice function does not include a check for potential failure scenarios when interacting with Chainlink oracles.

Recommendation:
Implement error handling mechanisms to handle potential failures in fetching oracle data.

Client comment: If the Chainlink price fetching is down temporarily, it will be reverted and is as intended. However, in the case there is another reason the Chainlink service cannot be used, such as a chance in API service, the service can be continuously operation after updating the Oracle contract.
In the future, if an error occurs in the implemented method (such as change in price fetching method from Chainlink, or service interruption from Chainlink), it will be updated so that another method can be used.

LOW-2 | RESOLVED

2-step ownership transfer

In Contract VaultETH_V2, method `transferOwnership()` allows the current DEFAULT_ADMIN_ROLE to set a new DEFAULT_ADMIN_ROLE. If the wrong address is set for the `newOwner`, it will be irreversible.
Instead, opt for a 2-step ownership transfer. In the first step, the current DEFAULT_ADMIN_ROLE sets the pendingNewOwner followed by pendingNewOwner accepting the ownership.

Recommendation:
Update the ownership transfer to 2-step process as suggested.

LOW-3 | ACKNOWLEDGED

calcFee Would Return Incorrect Value For A Fee-On-Transfer targetToken

Inside calcFee function of VaultETH_V2 contract if the targetToken is a fee on transfer token then at the transfer on L263 , tokens less than ratioFee + fixedFee would be transferred . In this case the return value ratioFee + fixedFee would be incorrect since lesser tokens were transferred.
Client comment: We are not going to use a fee-on-transfer tokens.

LOW-4 | RESOLVED

No check for address(0)

In Contract VaultETH_V2, the method transferOwnership(address newOwner) does not check if the `address newOwner` is address(0) or not. If it is set by mistake then it will be irreversible.

In Contract VaultETH_V2, the method UpdateRiskManager(...) does not check if `address to` is address(0) or not. If set by mistake, the Risk Manager ratio will be set to address(0).

Recommendation:
Update the above methods to add validations for address(0) check.

LOW-5 | RESOLVED

Use .call() for transferring ETH

In Contract VaultETH_V2, several instances are using payable(...).transfer(...) for sending native tokens to addresses.
This uses a fixed 2300 gas and gas repricing may break this leading to execution fees not being refunded and being stuck in the contract forever resulting in loss of funds.

Recommendation:
Use .call() instead to transfer native tokens with proper reentrancy mitigation.

LOW-6 | RESOLVED

Missing mechanism to track Votes

There should be a mechanism to track votes for each round of decision. Otherwise it can result in incorrect consensus being calculated in new rounds of decision making. For example, let's say there are 3 users of the Multisig- Alice, Bob and Eve. Let's say in round 1, Alice casted true, Bob casted true and Eve casted false via the vote() function. Now let's say if the numConfirmationsRequired is 2, then the function getStatus() will return true. But let's say now that a new decision is to be made and a new round of Voting starts. This time everyone Votes in the same way except Bob who forgets to vote. In the case the votestatuses of Alice, Bob and Eve should have been true, false and false respectively. But instead it is now true, true and false respectively. This would result in getStatus() again returning as true instead of false. This can lead to inconsistent or incorrect decisions being made.

Recommendation:

It is advised to review business and operational logic and introduce a mechanism to track votes for each round of decision being made. And then reset the voting status before each new round of voting.

LOW-7 | RESOLVED

Typing error in the RemoveContract_toStableToken() function

In the MultiSig_V2 contract, the line: 145 in the RemoveContract_toStableToken() function is as follows:

```
    IStalbeToken(StableToken).RemoveVaultManager(_target);(_target);
```

The code **"(_target);"** has been repeated twice in the function as an error and is not necessary. This could render confusion and render the contract undeployable.

Recommendation:

It is advised to remove the duplication of **(_target);**

LOW-8 | RESOLVED

Transfer ownership to zero address in RiskManagerEscrow_V2

The transferOwnership function can be used to accidentally transfer ownership of the contract to a zero address and revoke the role of the current owner. This would lead to onlyRole(DEFAULT_ADMIN_ROLE) function uncallable.

Recommendation:
To avoid this it is advised to add a zero address check for the newOwner parameter of the function.

LOW-9 | RESOLVED

Missing zero value check for gap in RiskManagerEscrow_V2

There is no check in the setPara() function to check that the _ratio parameter assigned to the gap variable should not be zero. A zero value assigned to gap can lead to division by zero panic on line: 172. Once the gap amiable is assigned, it cannot be changed again.

```
        return riskManagerEscrowAmt[riskManager][escrowToken] * (10 ** deltaDecimal) *
unitRatio / gap;
```

Recommendation:
It is advised to add a zero value check for the gap variable.

LOW-10 | RESOLVED

Unsafe Downcasting in TokenStableV6_V2

There is unsafe downcasting on line: 483 and 492.
Line: 483	uint16 currentEpoch = uint16((block.number - epochStart) / epochTerm + 1);

Line: 492	return uint16((block.number - epochStart) / epochTerm + 1);
This can result in silent overflows or truncation of the number.

Recommendation:
It is advised to use a safecast library such as that of Openzeppelin's Safecast library.

LOW-11 | RESOLVED

MIssing sanity value checks in the constructor of SwapOne2One

In the **SwapOne2One** contract, there is missing sanity checks for parameters of the constructor which are- _tokenA, _tokenB and _feePercentage.

Recommendation:
It is advised to add appropriate sanity value checks as follows:
1. Add a zero address check for _tokenA and _tokenB
2. Add a zero value check for _feePercentage and an upper value check or limit such as the one in setFeePercentage() function on line: 134.

LOW-12 | ACKNOWLEDGED

Add openzeppelin's safeMath Library in StableSwap_V2 and VaultETH_V2

Arithmetic operations in this contract does not use openzeppelin safeMath library to perform operations.

Recommendation:
Utilize the SafeMath library to prevent overflow and underflow vulnerabilities for all arithmetic operations in this contract.

Client comment: Solidity 0.8.0 onwards, it is provided by default. Also, the safeMath lib has disappeared starting with Openzeppelin 5.0. Still, we would appreciate it if you could advise on how to apply it as necessary.

LOW-13 | RESOLVED

Transfers To The Risk Managers Might Revert

A user can sell his ETH for stable tokens using the SellETH function in VaultETH_V2 , it is possible that no ETH is left in the call/contract to compensate for the risk managers at L318 , in that case the transfer would revert.

Recommendation:
Have a require statement to make sure there is enough ETH in the contract.

LOW-14 | RESOLVED

Use Of slot0 To Get sqrtPriceX96 Can Lead To Price Manipulation

Inside OracleLibrary the price (sqrtPriceX96) from UniswapV3 AMM is pulled from slot0, which is the most recent data point and can be manipulated easily via MEV bots and flashloans with sandwich attacks.
Though this is handled via the price deviation checks, the results can still be manipulated to an extent.

Recommendation:
Use TWAP price instead.

LOW-15 | ACKNOWLEDGED

Withdraw Function Might Revert If Not Sufficient Balance Of Tokens

A liquidity provider can withdraw his tokens using the withdraw() function in the SwapOneToOne contract. If the logic enters the else if at L111 where there is not enough tokenA balance in the contract , then the deficit of tokenA is compensated through tokenB transfer at L114 . But if there was not enough tokenB in the contract the transfer at L114 would revert , same reasoning for else-if condition at L117.

Recommendation:

Add a require statement to make sure the token balances is sufficient.

INFORMATIONAL-1 | RESOLVED

Manipulate112 Should Manipulate To uint112 Instead Of 128

The function manipulate112 manipulates a int128 whereas it should be applied to int112 instead

Recommendation:

Cast to a int112 instead

INFORMATIONAL-2 | RESOLVED

Missing checks for parameters for setParams Function in ERC20PriceOracle_V2

The setParams function allows the owner to set parameters, including unitRatio, gap, and precisionLimit.

Recommendation:

Ensure that these parameters are set within reasonable and safe ranges to avoid potential vulnerabilities.

INFORMATIONAL-3 | RESOLVED

Missing check for array length greater than 0 getERC20Price_High and getERC20Price_Low Functions

In both functions, there is a loop over the AMMInfos array to calculate the average price

Recommendation:

Consider adding a check to ensure that the AMMInfos array is not empty before entering the loop to prevent division by zero.

INFORMATIONAL-4 | RESOLVED

CEI not followed in WithdrawEscrow()

Checks effects interactions pattern has not been followed in the withdrawEscrow() function of the RiskManagerEscrow contract. As a result a compromised admin may be able carry out a reentrancy attack on the contract.

Recommendation:

It is advised to follow checks-effects interactions pattern as a best practice for the same.

INFORMATIONAL-5 | ACKNOWLEDGED

Missing checks in CheckRiskManagerStatus function

This assumes that the inputs (escrow, riskManager, inputAmt, tokenPrice, tokenPriceDecimal) are valid and within reasonable ranges.

Recommendation:

Consider including input validation checks to ensure that these values are as expected.

INFORMATIONAL-6 | RESOLVED

Unused params

In Contract VaultETH_V2, the method mintableAmount() comments mention that it returns the mintable amount based on user and epoch value.
Although it doesn't use `address user` anywhere in calculating the same.

Recommendation:
Either update the comment or the method as it suits the protocol.

INFORMATIONAL-7 | RESOLVED

Repetitive condition check

In Contract VaultETH_V2, the method Redeem_BuyETH(...) first asserts the following:

```
userWithdrawRequest.requestWithdrawAmt <= address(this).balance
```

Later on, check the same in an `if` condition which is not needed as it is already asserted to be true and no ETH transfers happened between those lines.

Recommendation:
Remove this check from the `if` condition.

INFORMATIONAL-8 | RESOLVED

Events parameters not indexed

In Contract VaultETH_V2, none of the events has `indexed` parameters to ease out the filter of event logs. Indexing seller and buyer parameters in the Sell and Buy events, respectively, is advised.

Recommendation:
Update the events to index the suggested parameters.

INFORMATIONAL-9 | RESOLVED

No validation when setting parameters

In Contract VaultETH_V2, methods setParam(...) and setParams2(...) set a lot of different values but they are not validated.

Recommendation:
Add validation for setup parameters to ensure proper configuration values.

INFORMATIONAL-10 | RESOLVED

CEI not followed in WithdrawEscrow()

Checks effects interactions pattern has not been followed in the withdrawEscrow() function of the RiskManagerEscrow contract. As a result a compromised admin may be able carry out a reentrancy attack on the contract.

Recommendation:
It is advised to follow checks-effects interactions pattern as a best practice for the same.

INFORMATIONAL-11 | RESOLVED

Floating pragma

Audited contracts use the following floating pragma:

```
pragma solidity ^0.8.19;
```

It allows to compile contracts with various versions of the compiler and introduces the risk of using a different version when deploying than during testing.

Recommendation:
Use a specific version of the Solidity compiler.

INFORMATIONAL-12 | RESOLVED

Typos and Incorrect comment

In Contract VaultETH_V2, correct the typo error for the mapping riksManagerList, it should be riskManagerList.

In Contract VaultETH_V2, mapping userAmountAtEpoch has an incorrect comment.

Recommendation:
Update the suggested changes.

INFORMATIONAL-13 | ACKNOWLEDGED

The SwapOne2One_V2 pool assumes token value to be constant

The pool follows an X = Y linear curve for swaps. Nobody will want to put the costlier token in the pool as a liquidity provider. This is because let's say the market price of token A is $5 and token B is $2. Now in the pool there are 1000 tokens A and 1000 tokens B. A user would immediately swap from token B to token A. This is because he would be getting a profit straightaway as he would be getting almost the same amount of tokens he deposits(minus fees). So let's say that a user swaps B for A for 500 tokens. Assuming the fees as negligible, the user would be profiting 500*5 - 500*2 = $1500. Now the pool will have 500 token A and 1500 token B (assuming fees to be negligible for simpler calculation). As the pool follows an X = Y curve, the pool actually always assumes that the price of token A and token B are the same. This would again result in draining of token A as its actual market price is higher than token B, and thus users would be getting token A at a discount. The pool will soon end up in place where there will be 0 token A and 2000 token B. Thus, no liquidity provider would want to provide token A as he would be losing the costlier token.
Thus this pool will soon lead to liquidity issues and become defunct quickly.

The amount to be swapped in this pool remains in a 1:1 ratio. The swappable amount from one token to the other always remains constant. This can be detrimental if the token depegs from its value. It would again result in the situation as discussed above and lead to market manipulation and liquidity crisis.

Recommendation:
It is advised to not use tokens for this pool which are not pegged 1:1 with each other.

INFORMATIONAL-14 | RESOLVED

RiskManagerEscrow Cannot Receive Ether

The transfer at L90 in RiskManagerEscrow_V2 transfers ETH in the contract to the admin , but , there is no receive/fallback or payable function to receive Ether

Recommendation:
Since there can never be ether in the contract (unless it is forced sent through self destruct) the logic to transfer eth can be removed.

INFORMATIONAL-15 | RESOLVED

Directly Calling The calcFee Function Will Result In Lost Ether

Directly calling the calcFee function in VaultEth_V2.sol will result in loss of ETH for the caller , since the function calculates the fee and deducts the fee from the user.

Recommendation:
Make the function only callable by the contract itself.

INFORMATIONAL-16 | RESOLVED

Unnecessary Assignment

The assignment tokenBBalance and tokenABalance in the constructor of SwapOneToOne assigns them to the balance of each token , since during deployment the balance of each would be 0 , it does not make sense to query the balanceOf() for the tokens , just a simple declaration would do suffice.

Recommendation:
No need to assign to balanceOf(token)

	./Contract_V002/interface/IStable.sol ./Contract_V002/interface/IRiskManagerEscrow.sol ./Contract_V002/interface/IEscrowToken.sol ./Contract_V002/interface/IUniswapV3Pool.sol ./Contract_V002/interface/IERC20PriceOracle.sol ./Contract_V002/interface/IVaultManager.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

	./Contract_V002/interface/IUniswapV2Pair.sol ./Contract_V002/library/Type.sol ./Contract_V002/library/RiskManagerEscrowLibrary.sol ./Contract_V002/library/OracleLibrary.sol ./Contract_V002/library/Utils.sol ./Contract_V002/MultiSig_V2.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

	./Contract_V002/RiskManagerEscrow_V2.sol ./Contract_V002/ERC20PriceOracle_V2.sol ./Contract_V002/TokenStableV6_V2.sol ./Contract_V002/SwapOne2One_V2.sol ./Contract_V002/VaultETH_V2.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 TRZN 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 TRZN team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.