



Security Assessment

TIVEL FINANCE

Verified On July 12th, 2024

@Vital-Block

@VB_Audit

info@vitalblock.org

www.vitalblock.org



PREPARED FOR:
TIVEL FINANCE






TABLE OF CONTENTS

TABLE OF CONTENTS	3
DOCUMENT PROPERTIES	4
ABOUT VBS	5
SCOPE OF WORK	6
AUDIT METHODOLOGY	7
AUDIT CHECKLIST	9
EXECUTIVE SUMMARY	10
CENTRALIZED PRIVILEGES	11
RISK CATEGORIES	12
AUDIT SCOPE	13
AUTOMATED ANALYSIS	14
KEY FINDINGS	19
MANUAL REVIEW	20
VULNERABILITY SCAN	28
REPOSITORY	29
INHERITANCE GRAPH	30
PROJECT BASIC KNOWLEDGE	31
AUDIT RESULT	32
REFERENCES	37



INTRODUCTION

Auditing Firm	 VITAL BLOCK SECURITY
Client Firm	 TIVEL FINANCE
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	<p>Factory.sol</p> <p>PoolDeployer.sol</p> <p>Pool.sol</p> <p>WithdrawalMonitor.sol</p> <p>PositionStorage.sol</p> <p>PriceFeed.Sol</p> <p>DEXAggregatorV2.sol</p> <p>LiquidationMaker.sol</p> <p>MetaAggregator.sol</p> <p>Monitor.sol</p> <p>NonfungiblePositionManager.sol</p>
Network	 Zksync
Centralization	Active ownership
Website	https://tivel.finance/
Discord	https://discord.com/invite/zxaH2u9dXM
Twitter	https://x.com/tivelfinance
GitHub	https://github.com/tivelprotocol/
Prelim Report Date	July 10 th , 2024
Final Report Date	July 12 th 2024



Verify the authenticity of this report on our GitHub Repo: <https://www.github.com/vital-block>



Document Properties


Client	TIVEL FINANCE
Title	Smart Contract Audit Report
Target	TIVEL FINANCE
Audit Version	1.2
Author	Akhmetshin Marat
Auditors	Akhmetshin Marat, James BK, Benny Matin
Reviewed by	Dima Meru
Approved by	Prince Mitchell
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.2	July 11 th , 2024	James BK	Final Released
1.2-AP	July 12 th , 2024	Benny Matin	Release Candidate

Contact

For more information about this document and its contents, please contact Vital Block Security Inc.

Name	Akhmetshin Marat
Phone 	+44 7944 248057
Email	info@vitalblock.org

In the following, we show the specific pull request and the commit hash value used in this audit.

- <https://github.com/tivprotocol/tivel-contracts-zk/tree/master/contracts> (TIV-72761)
- <https://github.com/tivprotocol/tivel-contracts-zk/blob/master/contracts/Pool.sol> (TIVU144210)

About Vital Block Security

Vital Block Security provides professional, thorough, fast, and easy-to-understand smart contract security audit. We do in-depth and penetrative static, manual, automated, and intelligent analysis of the smart contract. Some of our automated scans include tools like ConsenSys MythX, Mythril, Slither, Surya. We can audit custom smart contracts, DApps, Rust, NFTs, etc (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/vital_block), Twitter (http://twitter.com/Vb_Audit), or Email (info@vitalblock.org).

Table 1.2: Vulnerability Severity Classification

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

Methodology (1)

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [4]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

SCOPE OF WORK

Vital Block was consulted by **TIVEL FINANCE** to conduct the smart contract audit of its. Move source code. The audit scope of work is strictly limited to mentioned .SOL file only:

TIVEL.FINANCE

i External contracts and/or interfaces dependencies are not checked due to being out of scope.

Verify audited contract's contract address and deployed link below:

Public Contract Code Audited

Factory.sol

PoolDeployer.sol

Pool.sol

WithdrawalMonitor.sol

PositionStorage.sol

PriceFeed.Sol

DEXAggregatorV2.sol

LiquidationMaker.sol

MetaAggregator.sol

Monitor.sol

NonfungiblePositionManager.sol

Project Name:

TIVEL FINANCE



Table 1.0 The Full Audit Checklist

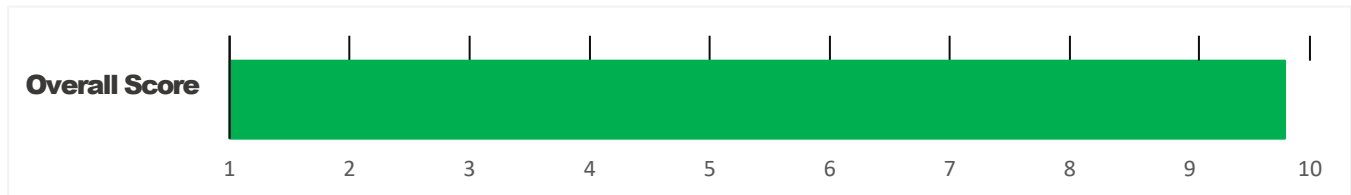
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

EXECUTIVE SUMMARY

Vital Block Security has performed the automated and manual analysis of the **TIVEL FINANCE** Sol code. The code was reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical ! 🔴	Major " 🟡	Medium # 🟡	Minor \$ 🟢	Unknown % 🟤
Open	0	0	0	2	0
Acknowledged	0	0	3	3	6
Resolved	0	0	0	0	0
Noteworthy onlyOwner Privileges	Set Taxes and Ratios, Airdrop, Set Protection Settings, Set Reward Properties, Set Reflector Settings, Set Swap Settings, Set Pair and Router				

FUTURE FINANCE Smart contract has achieved the following score: **%98.5**



i Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

i Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.



AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of Vital Block auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
 - Remix IDE Developer Tool
 - Open Zeppelin Code Analyzer
 - SWC Vulnerabilities Registry
 - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges.

We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	<ul style="list-style-type: none">○ Token Supply Manipulation○ Access Control and Authorization○ Assets Manipulation○ Ownership Control○ Liquidity Access○ Stop and Pause Trading○ Ownable Library Verification
----------------------	---



Common Contract Vulnerabilities

- **Integer Overflow**
- **Lack of Arbitrary limits**
- **Incorrect Inheritance Order**
- **Typographical Errors**
- **Requirement Violation**
- **Gas Optimization**
- **Coding Style Violations**
- **Re-entrancy**
- **Third-Party Dependencies**
- **Potential Sandwich Attacks**
- **Irrelevant Codes**
- **Divide before multiply**
- **Conformance to Solidity Naming Guides**
- **Compiler Specific Warnings**
- **Language Specific Warnings**

REPORT

- **The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.**
- **The client's development team reviews the report and makes amendments to the codes.**
- **The auditing team provides the final comprehensive report with open and unresolved issues.**

PUBLISH

- **The client may use the audit report internally or disclose it publicly.**

 **It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.**



RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical 🚨	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major 🟡	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium 🟠	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.



CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- **Privileged roles can be granted the power to `pause()` the contract in case of an external attack.**
- **Privileged roles can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.**

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- **The client can lower centralization-related risks by implementing below mentioned practices:**
- **Privileged role's private key must be carefully secured to avoid any potential hack.**
- **Privileged role should be shared by multi-signature (multi-sig) wallets.**
- **Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.**
- **Renouncing the contract ownership, and privileged roles.**
- **Remove functions with elevated centralization risk.**

 Understand the project's initial asset distribution. Assets in the liquidity pair should be locked.

Assets outside the liquidity pair should be locked with a release schedule.



Key Findings

Overall, these contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table [2.1](#)), 0 medium-severity vulnerabilities, 3 low-severity vulnerabilities, and 1 informational recommendations.

Table 2.1: Key **TIVEL FINANCE** Audit Findings

ID	Severity	Title	Category	Status
TTY-01	Informational	In UncheckedForTransfer , the following equation is used inside an unchecked block	Status Mathematical Operations	Acknowledged
TNY-02	Medium	In updateForOwner, Relevant Function Snippet	Business Logic	Acknowledged
TDL-03	Inconsistency	In Suggested Constant/Immutable Usages For Gas Efficiency	Coding Practice	Acknowledged
TXL-04	Low	Asset Price in GenericLogic: collateralAccount()	GenericLogic	Acknowledged
TEL-05	Pool	Proper EMode Category Use in Pool::borrow()	Coding Practice	Acknowledged
TQL-06	BorrowLogic	Possible Underflow Avoidance in BorrowLogic And UserConfiguration	Coding Practice	Acknowledged

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to page [10](#) for details...






AUDIT SCOPE

TIVEL FINANCE

ID	Repo	Comment	File	SHM211 Checksum
TTM	Tivel-contracts-zk/tree/master/contracts	Cc51d21	Factory.sol	85f15802c6be0fd50f8632d8433cccc9db6f4b39f9e566d1fa78de54b84bddr54
TRY	Tivel-contracts-zk/tree/master/contracts	cC51D32	Factory.sol	8oippkjjk96be0fd50f8632d8433cccc9db6f4b39f9e566d1yhhg8765ffckuiybb
TTV	Tivel-contracts-zk/tree/master/contracts	cC51D42	PoolDeployer.sol	3666778uj908766362fvyga98jdkl88648yhfbqt37409owehbgwhuyyyg223738
TML	Tivel-contracts-zk/tree/master/contracts	cC51D44	PoolDeployer.sol	98uuyriy399787390uhbiiuhghhdg7guu30oi7799u9359ydfgdgygeigi3ioueyy78
TTR	Tivel-contracts-zk/tree/master/contracts	cC51D46	Pool.sol	4566efgywqtfeuh87872t15378837983639293763hhegetgjfwjk89336668862
TOP	Tivel-contracts-zk/tree/master/contracts	cC51D48	Pool.sol	546363ttebnve88329973mvvdsggct478153ytdgfdxy792635fgdjgi1900990908
TDP	Tivel-contracts-zk/tree/master/contracts	cC51D49	WithdrawalMonitor.sol	835656990327hubbinnjnr6729dchjld0993ytyy3vq63235727879889073
TWY	Tivel-contracts-zk/tree/master/contracts	cC51D53	WithdrawalMonitor.sol	cc089692343d1cc36eaf196046d7a528d153abd55ba20e82f1d57c22fcd92675
TKB	Tivel-contracts-zk/tree/master/contracts	cC51D62	PositionStorage.sol	8448b3af42497f5f74e53424ee3e6c551f51356945108d22a893d608a7990542
TXY	Tivel-contracts-zk/tree/master/contracts	cC51D63	PriceFeed.sol	5c86aa1dd3889db5fcd17a80214b226fc784f268ab9db82df97c1d2459467831
TCB	Tivel-contracts-zk/tree/master/contracts	cC51D63	DEXAggregatorV2.sol	b8244da33db171e5533d77bef4a35703df1de2cebea5f35cb38ce6a26c778cf1
TWO	Tivel-contracts-zk/tree/master/contracts	cC51D67	LiquidationMarker.sol	3d408b8f2cc56f9699a402b5151de90671de089c3007afc9e4fc867c04152e7c
TGT	Tivel-contracts-zk/tree/master/contracts	cC51D68	MetaAggregator.sol	9d751621c3501102e4b50005ca3314ec6e04e6ff8bbb30852d1c7edfff3f8cef
TDF	Tivel-contracts-zk/tree/master/contracts	cC51D72	Monitor.sol	455687gfsadjknlpuiuhg774580vgfxrki9876dhgvb990lkjhde444566788
TGV	Tivel-contracts-zk/tree/master/contracts	cC51D85	NonfungiblePositionManager.sol	hbgyyyutwi7653896793jfohjklnfnwwwqafenoggyueppjete38543



AUTOMATED ANALYSIS

Symbol	Definition
	Function modifies state
	Function is payable
	Function is internal
	Function is private
	Function is important

```

**TIVEL FINANCE** | Interface | |||
| L | totalSupply | External ! | ! | NO ! |
| L | decimals | External ! | ! | NO ! |
| L | symbol | External ! | ! | NO ! |
| L | name | External ! | ! | NO ! |
| L | getOwner | External ! | | NO ! |
| L | balanceOf | External ! | ! | NO ! |
| L | transfer | External ! | " ! ! | NO ! |
| L | allowance | External ! | ! | NO ! |
| L | approve | External ! | " ! ! | NO ! |
| L | transferFrom | External ! | " | NO ! |
|||||
**IFactoryV2** | Interface | |||
| L | getPair | External ! | | NO ! |
| L | createPair | External ! | " | NO ! |
|||||
**IV2Pair** | Interface | |||
| L | factory | External ! | | NO ! |
| L | getReserves | External ! | | NO ! |
| L | sync | External ! | " | NO ! |

```



|||||

| ****IRouter01**** | Interface | |||

| L | factory | External ! | |NO!|

| L | SOL | External ! | |NO!|

| L | addLiquidityETH | External ! | # |NO!|

| L | addLiquidity | External ! | " |NO!|

| L | swapExactETHForTokens | External ! | # |NO!|

| L | getAmountsOut | External ! | |NO!|

| L | getAmountsIn | External ! | |NO!|

|||||

| ****IRouter02**** | Interface | IRouter01 |||

| L | swapExactTokensForAPTSupportingFeeOnTransferTokens | External ! | " |NO!|

| L | swapExactETHForTokensSupportingFeeOnTransferTokens | External ! | # |NO!|

| L | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ! | " ! |NO!|

| L | swapExactTokensForTokens | External ! | " |NO!|

|||||

| ****Protections**** | Interface | |||

| L | checkUser | External ! | " ! |NO!|

| L | setLaunch | External ! | " |NO!|

| L | setLpPair | External ! | " |NO!|

| L | **ETH** | External ! | " |NO!|

| L | removeSniper | External ! | " |NO!|

|||||

| ****Cashier**** | Interface | |||

| L | setRewardsProperties | External ! | " |NO!|

| L | tally | External ! | " |NO!|

| L | load | External ! | # |NO!|

| L | cashout | External ! | " |NO!|

| L | giveMeWelfarePlease | External ! | " |NO!|

| L | getTotalDistributed | External ! | |NO!|

| L | getUserInfo | External ! | |NO!|

| L | getUserRealizedRewards | External ! | |NO!|




```

| L | getPendingRewards | External ! | | NO! |
| L | initialize | External ! | " | NO! |
| L | getCurrentReward | External ! | | NO! |
|||||
| **SOL** | Implementation | SafeMath |||
| L | <Constructor> | Public ! | # | NO! |
| L | transferOwner | External ! | " | onlyOwner |
| L | renounceOwnership | External ! | " | NO! |
| L | setOperator | Public ! | " | NO! |
| L | renounceOriginalDeployer | External ! | " | NO! |
| L | <Receive ETH> | External ! | # | NO! |
| L | totalSupply | External ! | | NO! |
| L | decimals | External ! | | NO! |
| L | symbol | External ! | | NO! |
| L | name | External ! | | NO! |
| L | getOwner | External ! | ! | NO! |
| L | balanceOf | Public ! | ! | NO! |
| L | allowance | External ! | ! | NO! |
| L | approve | External ! | " ! 🚫 | NO! |
| L | _approve | Internal $ | " 🚫 🚫 ||
| L | approveContractContingency | Public ! | " ! 🚫 | onlyOwner |
| L | transfer | External ! | " | NO! |
| L | transferFrom | External ! | " | NO! |
| L | setNewRouter | External ! | " | onlyOwner |
| L | setLpPair | External ! | " | onlyOwner |
| L | setInitializers | External ! | " | onlyOwner |
| L | isExcludedFromFees | External ! | | NO! |
| L | isExcludedFromDividends | External ! | | NO! |
| L | isExcludedFromProtection | External ! | | NO! |
| L | setDividendExcluded | Public ! | " | onlyOwner |
| L | setExcludedFromFees | Public ! | " | onlyOwner |

```



TTY-01 Key Findings

Category	Severity ●	Location	Status
Status Mathematical Operations	Low	Contract/Pool.sol – 230-237	Informational

Description

In **UncheckedForTransfer**, the following equation is used inside an unchecked() block

```
function _transferProtocolFee() internal {
    uint256 _protocolFee = accProtocolFee;
    if (_protocolFee > 0) {
        address feeTo = IFactory(factory).protocolFeeTo();
        if (feeTo != address(0)) {
            accProtocolFee = 0;
            TransferHelper.safeTransfer(quoteToken, feeTo, _protocolFee);
        }
    }
}
```

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the **TivelFinance** contract.

Recommendation

Incorporate the following verification within process approve account to confirm that the contract account's associated transfer aligns with the mint for which the confidential transfer approval is sought.



TNY-02 Key Findings

Category	Severity ●	Target	Status
Business Logic	Medium	Contract/Positionstorage.sol -- 173-183	Low

Description

In **updateForOwner**, Relevant Function Snippet

```
function claimableFee(
    address _owner
) external view override returns (uint256 amount) {
    LiquidityPosition memory pos = liquidityPosition[_owner];
    amount =
        pos.pendingFee +
        (accFeePerShare * pos.liquidity) /
        precision -
        pos.feeDebt;
    if (amount > accFee) amount = accFee;
}
```

Description

For Ownership efficiency, the TIVELFINANCE Team is engineered with the reserve cache mechanism, which necessitates the common steps to be followed when operating with the reserve Ownership data in different scenarios, including the fee generation, update, and eventual persistence.

Recommendation

Revise the above functions to following a consistent approach to use the reserve cache mechanism.

TDL-03 Key Findings

Category	Severity ●	Location	Status
Coding Practices	Low	Contracts/withdrawalmonitor.sol – 198-215	Informational

Description

In Suggested Constant/Usages For Gas Efficiency

```
function performUpkeep(bytes calldata _performData) external override {
    uint256 usedGas = gasleft();
    (address[] memory poolsToFulfill, uint256 count) = abi.decode(
        _performData,
        (address[], uint256)
    );

    for (uint256 i = 0; i < count; i++) {
        address pool = poolsToFulfill[i];
        _execute(pool);
    }

    emit ProcessBatch(
        count,
        _performData,
        usedGas - gasleft(),
        tx.gasprice
    );
}
```

Description

Since version v0.8.15+, [Solidity](#) introduces the feature of declaring a state as `immutable`. An `immutable` state variable can only be assigned during contract creation, but will remain constant throughout the life-time of a deployed contract. The main benefit of declaring a state as `immutable` is that reading the state is significantly cheaper than reading from regular storage, since it is not stored in storage anymore. Instead, an `immutable` state will be directly inserted into the runtime code.

Recommendation

Revisit the state variable definition and make extensive use of `constant`/ Usage For Gas Efficiency.

TXL-04 Key Findings

Category	Severity ●	Target	Status
Coding Practices	Low	Contract/positionstorage.sol – 618-628	Informational

Description

Asset **Price in GenericLogic: collateralAmount()**

```
if (_params.updater != pos.owner)
    revert NotOwner(pos.owner, _params.updater);
IFactory _factory = IFactory(factory);
uint256 pricePrecision = PRICE_PRECISION;
uint256 baseTokenMUT = _factory.baseTokenMUT(pos.baseToken.id);
uint256 collateralMUT = _factory.collateralMUT(pos.collateral.id);
uint256 newCollateralAmount = pos.collateral.amount + _params.amount;

uint256 mutb = (pos.baseToken.amount *
    pos.baseToken.entryPrice *
    baseTokenMUT) / (pricePrecision * 10000);
```

Description

For any lending protocol, there is a need to reliably and accurately measure the borrower's debt position and provide necessary means to liquidate underwater positions. The TIVEL FINANCE protocol is no exception. While reviewing the implementation to measure the debt position, we notice the key function `collateralAmount()` needs to be improved...

Recommendation

Apply the right price oracle in the above `newcollateral.Amount` routine to compute the user account data.

THL-5 Key Findings

Category	Severity ●	Target	Status
Coding Practices	low	Contract/Positionstorage.sol –200-214	Informational

Description

Possible **Underflow Avoidance in BorrowLogic And UserConfiguration**

```
function setUpdateCollateralAmountFee(
    uint256 _updateCollateralAmountFee
) external onlyManager {
    updateCollateralAmountFee = _updateCollateralAmountFee;

    emit SetUpdateCollateralAmountFee(_updateCollateralAmountFee);
}

function setUpdateDeadlineFee(
    uint256 _updateDeadlineFee
) external onlyManager {
    updateDeadlineFee = _updateDeadlineFee;

    emit SetUpdateDeadlineFee(_updateDeadlineFee);
}
```

Description

The TIVEL FINANCE protocol has established itself as one of the leading lending protocol. Within each lending protocol, there is a constant need of accommodating various precision issues. SafeMath is a widely-used Solidity math library that is designed to support safe math operations by preventing common overflow or underflow issues when working with uint256 operands. Since the version 0.8.15+, Solidity includes checked arithmetic operations by default, and this largely renders SafeMath unnecessary. While re-viewing arithmetic operations in current implementation, we notice occasions that may introduce unexpected overflows/underflows.

For example, if we examine the updateCollateralAmountFee() function, it may revert if the current c updateDeadlineFee In (line 211) is equal to 0.

Recommendation

Revise the above calculation to avoid the unnecessary overflows and under- flows.

TQL-6 Key Findings

Category	Severity ●	Target	Status
Coding Practices	low	Contract/Factory.sol – 298-305	Fixed

Description

Possible **Underflow Avoidance in ReserveLogic And UserConfiguration**

```
function setPoolMaxBaseReserve(
    address _quoteToken,
    address _baseToken,
    uint256 _maxBaseReserve
) external onlyManager {
    address pool = poolByQuoteToken[_quoteToken];
    if (pool == address(0)) revert PoolNotExists(_quoteToken);
    IPool(pool).setMaxBaseReserve(_baseToken, _maxBaseReserve);
}
```






Description

For gas efficiency, the TIVEL FINANCE protocol is engineered with the reserve cache mechanism, which necessitates the common steps to be followed when operating with the reserve data in different scenarios, including the cache generation update and eventual persistence. However, our analysis shows certain inconsistency in the reserve cache usage.

Recommendation

We recommend either checking for overflow in this case, or ensuring that the PairsIn is close enough it will never cause an overflow. or use the reserve cache mechanism.

OPTIMIZATIONS | TIVEL FINANCE

ID	Title	Category	Status
FTV	Logarithm Refinement Optimization	Gas Optimization	Acknowledged 
FOP	Checks Can Be Performed Earlier	Gas Optimization	Acknowledged 
FDP	Unnecessary Use Of SafeMath	Gas Optimization	Acknowledged 
FWY	Struct Optimization	Gas Optimization	Acknowledged 
FGT	Unused State Variable	Gas Optimization	Acknowledged 

General Detectors

Missing Zero Address Validation

Some functions in this contract may not appropriately check for zero addresses being used.



Attention
Required

Incorrect Solidity Version

This contract uses an unconventional or very old version of Solidity



Attention
Required

- ✓ No compiler version inconsistencies found
- ✓ No unchecked call responses found
- ✓ No vulnerable self-destruct functions found
- ✓ No assertion vulnerabilities found
- ✓ No old solidity code found
- ✓ No external delegated calls found
- ✓ No external call dependency found
- ✓ No vulnerable authentication calls found
- ✓ No invalid character typos found
- ✓ No RTL characters found
- ✓ No dead code found
- ✓ No risky data allocation found
- ✓ No uninitialized state variables found
- ✓ No uninitialized storage variables found
- ✓ No vulnerable initialization functions found
- ✓ No risky data handling found
- ✓ No number accuracy bug found
- ✓ No out-of-range number vulnerability found
- ✓ No map data deletion vulnerabilities found
- ✓ No tautologies or contradictions found
- ✓ No faulty true/false values found
- ✓ No innacurate divisions found
- ✓ No redundant constructor calls found
- ✓ No vulnerable transfers found
- ✓ No vulnerable return values found
- ✓ No uninitialized local variables found
- ✓ No default function responses found
- ✓ No missing arithmetic events found
- ✓ No missing access control events found
- ✓ No redundant true/false comparisons found
- ✓ No state variables vulnerable through function calls found
- ✓ No buggy low-level calls found
- ✓ No expensive loops found
- ✓ No bad numeric notation practices found
- ✓ No missing constant declarations found
- ✓ No missing external function declarations found
- ✓ No vulnerable payable functions found
- ✓ No vulnerable message values found



Vulnerability Scan

REENTRANCY

✓ No reentrancy risk found

Severity Minor

Confidence Parameter Certain

Vulnerability Description

✗ **Not Mintable**: A large amount of this token can not be minted by a private wallet or contract.

Scanning Line:

```
function mint(  
    address _to,  
    uint256 _liquidity,  
    bytes calldata _data  
) external override lock {  
    address _quoteToken = quoteToken;  
    LiquidityPosition storage pos =  
    liquidityPosition[_to];  
  
    uint256 _accFeePerShare = accFeePerShare;  
    uint256 _precision = precision;
```

Identifier	Definition	Severity
CEN-02	Initial asset distribution	Minor 

```
uint256 balanceBefore = IERC20(_quoteToken).balanceOf(address(this));
IMintCallback(msg.sender).mintCallback(_quoteToken, _liquidity, _data);

uint256 balance = IERC20(_quoteToken).balanceOf(address(this));
if (balance < balanceBefore + _liquidity) revert InsufficientInput();

emit Mint(msg.sender, _to, _liquidity);
}
```

Description:

Floating point calculations can vary across different architectures.

Alleviation:

This exhibit was acknowledged and ultimately discarded by the **TIVEL FINANCE** team due to low severity. We consider the exhibit fully attended to as it doesn't impose any meaningful security concerns.

RECOMMENDATION

Project stakeholders should be consulted during the initial asset distribution process.



Repository:

<https://github.com/tivelprotocol/tivel-contracts-zk/tree/master/contracts>

All Audited Files

Factory.sol

PoolDeployer.sol

Pool.sol

WithdrawalMonitor.sol

PositionStorage.sol

PriceFeed.Sol

DEXAggregatorV2.sol

LiquidationMaker.sol

MetaAggregator.sol

Monitor.sol

NonfungiblePositionManager.sol

Contract Deployed

Contract :

Factory: NOT DEPLOYED

PoolDeployer: NOT DEPLOYED

Pool: NOT DEPLOYED

WithdrawalMonitor: NOT DEPLOYED

PositionStorage: NOT DEPLOYED

PriceFeed: NOT DEPLOYED

DEXAggregatorV2: NOT DEPLOYED

LiquidationMaker: NOT DEPLOYED

MetaAggregator: <https://era.zksync.network/address/0xD53d68eE30a0863B2922908e416E36cA0AC4B864#code>

Monitor: NOT DEPLOYED

NonfungiblePositionManager: NOT DEPLOYED



Vulnerability Run check

Risk Analysis

✔ Contract source code verified

This token contract is open source. You can check the contract code for details. Unsourced token contracts are likely to have malicious functions to defraud their users of their assets.

✔ No mint function

Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token.

✔ Owner cant change balance

The contract owner does not have the authority to modify the balance of tokens at other addresses.

✔ No Proxy

There is no proxy in the contract. The proxy contract means contract owner can modify the function of the token and possibly effect the price.

✔ No function to retrieve ownership

If this function exists, it is possible for the project owner to regain ownership even after relinquishing it.



Honeypot Risk

✔ This does not appear to be a honeypot

We are not aware of any code that prevents the sale of tokens.

✔ No Anti Whale

There is no limit to the number of token transactions. The number of scam token transactions may be limited (honeypot risk).

✔ No whitelist function

Whitelist function found

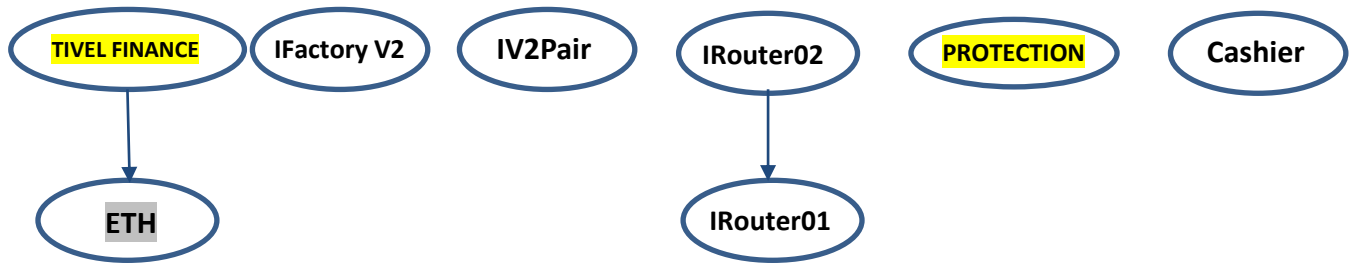
✔ No trading cooldown

The token contract has no trading cooldown function. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying.

✔ No blacklist function

No blacklist function is included.

INHERITANCE GRAPH



Identifier	Definition	TIVEL FINANCE
CEN-12	Centralization privileges of FUTURE FINANCE	Medium #

Vulnerability 0 : No important security issue detected.

Threat level: Low

```

272 IMintCallback(msg.sender).mintCallback(_quoteToken, _liquidity, _data);
273
274 uint256 balance = IERC20(_quoteToken).balanceOf(address(this));
275 if (balance < balanceBefore + _liquidity) revert InsufficientInput();
276
277 emit Mint(msg.sender, _to, _liquidity);
278 }
279
280 function collect(address _to, uint256 _amount) external override lock {
281     if (_amount > 0) {
282         LiquidityPosition storage pos = liquidityPosition[msg.sender];
283
284         uint256 _accFeePerShare = accFeePerShare;
285         uint256 _precision = precision;
286
287         if (pos.liquidity > 0) {
288             pos.pendingFee +=
289                 (_accFeePerShare * pos.liquidity) /
290                 _precision -
291                 pos.feeDebt;
292         }
293         pos.feeDebt = (_accFeePerShare * pos.liquidity) / _precision;
294     }
  
```

MANUAL REVIEW

The first lending-based DEX built on zkSync, allowing traders the flexibility to choose their desired trade price without relying on matching orders and being obligated to trade at the prevailing market price

Lending-based DEX is the NEXT GEN of spot trading, allowing traders the flexibility to choose their desired trade price without relying on matching orders and being obligated to trade at the prevailing market price. Traders receive the results of trades instantly from single-token liquidity pools. To achieve this, trading asset and collateral will be monitored before officially executing the trade and providing the final result at the time of closing.

TOKEN NAME: TIVEL FINANCE

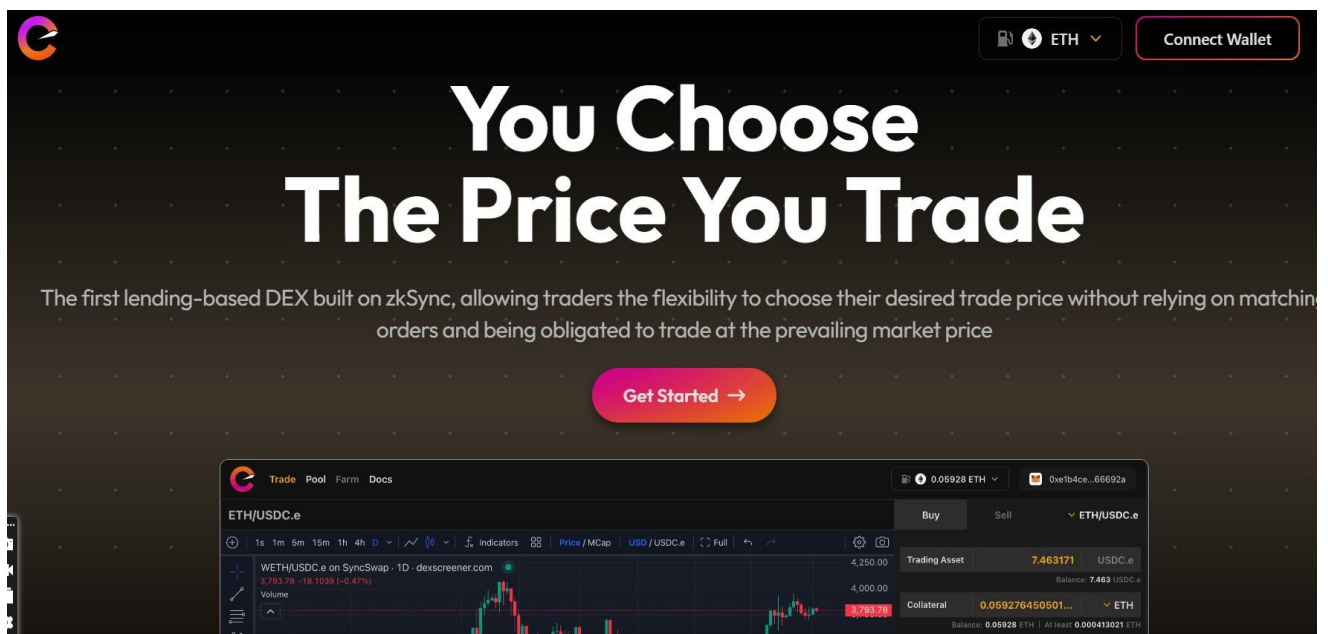
Gen: Protocol

Chain/Standard: zkSync

LAUNGUGE: Solidity



The TIVEL FINANCE Platform Is Launching On zkSync Chain





ISSUES CHECKING STATUS

Issue Description

Checking Status

1.	Compiler errors	PASSED
2.	Race Conditions and reentrancy. Cross-Function Race Conditions.	PASSED
3.	Possible Delay In Data Delivery.	PASSED
4.	Oracle calls.	PASSED
5.	Front Running.	PASSED
6.	Move Dependency.	PASSED
7.	Integer Overflow And Underflow.	PASSED
8.	DoS with Revert.	PASSED
9.	Dos With Block Gas Limit.	PASSED
10.	Methods execution permissions.	PASSED
11.	Economy Model of the contract.	PASSED
12.	The Impact Of Exchange Rate On the Move Logic.	PASSED
13.	Private use data leaks.	PASSED
14.	Malicious Event log.	PASSED
15.	Scoping and Declarations.	PASSED
16.	Uninitialized storage pointers.	PASSED
17.	Arithmetic accuracy.	PASSED
18.	Design Logic.	PASSED
19.	Cross-Function race Conditions	PASSED
20.	Save Upon Move contract Implementation and Usage.	PASSED
21.	Fallback Function Security	PASSED



AUDIT RESULT

PASSED

SMART CONTRACT AUDIT OF TIVEL FINANCE

Identifier	Definition	Severity
CEN-02	Initial asset distribution	Minor 

All of the initially minted assets are sent to the contract deployer when deploying the contract. This can be an issue as the deployer and/or contract owner can distribute tokens without consulting the community.

}

```
function getLowestPrice(
    address _baseToken,
    address _quoteToken
) external view override returns (uint256 lowest) {
    if (_baseToken == _quoteToken) {
        return _PRECISION;
```

RECOMMENDATION

Project stakeholders should be consulted during the initial asset distribution process.

RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully.

Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

ALLEVIATION

The TIVEL FINANCE project team understands the centralization risk. Some functions are provided privileged access to ensure a good runtime behavior in the project



Identifier	Definition	Severity
COD-10	Third Party Dependencies	Minor 

Smart contract is interacting with third party protocols e.g., Pancakeswap router, cashier contract, protections contract. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect and validate third party dependencies regularly, and mitigate severe impacts whenever necessary.



DISCLAIMERS

Vital Block provides the easy-to-understand audit of Solidity, Move and Raw source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. 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. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way



to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

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

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, 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, VITAL BLOCK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, VITAL BLOCK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, VITAL BLOCK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT’S OR ANY OTHER INDIVIDUAL’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.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. Vital Block does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.



LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than Vital Block. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites and social accounts owners. You agree that Vital block Security is not responsible for the content or operation of such websites and social accounts and that Vital Block shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.



ABOUT VITAL BLOCK

Vital Block provides intelligent blockchain Security Solutions. We provide solidity and Raw Code Review, testing, and auditing services. We have Partnered with 15+ Crypto Launchpads, audited 50+ smart contracts, and analyzed 200,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Aptos, Oasis, etc.

Vital Block is Dedicated to Making Defi & Web3 A Safer Place. We are Powered by Security engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 5 core members, and 4+ casual contributors.

Website: <https://Vitalblock.org>

Email: info@vitalblock.org

GitHub: <https://github.com/vital-block>

Telegram (Engineering): https://t.me/vital_block

Telegram (Onboarding): https://t.me/vitalblock_cmo





vital-block



info@vitalblock.org



www.Vitalblock.org



Vital Block Dedicated to securing Public and Private Blockchain Ecosystem