VITAL BLOCK

Security Assessment

TIVEL FINANCE

Verified On July 11th, 2024

@Vital-Block

@VB_Audit

info@vitalblock.org

www.vitalblock.org

PREPARED FOR:
TIVEL FINANCE

TABLE OF CONTENTS

INTRODUCTION

Auditing Firm	**VITAL BLOCK SECURITY**
Client Firm	**TIVEL FINANCE**
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	Factory.sol PoolDeployer.sol Pool.sol WithdrawalMonitor.sol PositionStorage.sol PriceFeed.Sol DEXAggregatorV2.sol LiquidationMaker.sol MetaAggregator.sol Monitor.sol NonfungiblePositionManager.sol
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	July10th , 2024
Final Report Date	July 11th 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	1.0
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
1.0	July 10th , 2024	James BK	Final Released
1.0-AP	July 11th , 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/tivelprotocol/tivel-contracts-zk/tree/master/contracts (TIV-72761)
- https://github.com/tivelprotocol/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 \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

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:

o.TIVELFINANCE.Sol

ℹ️ **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	
Contract Name	**TIVEL FINANCE**

Table 1.0 The Full Audit Checklist

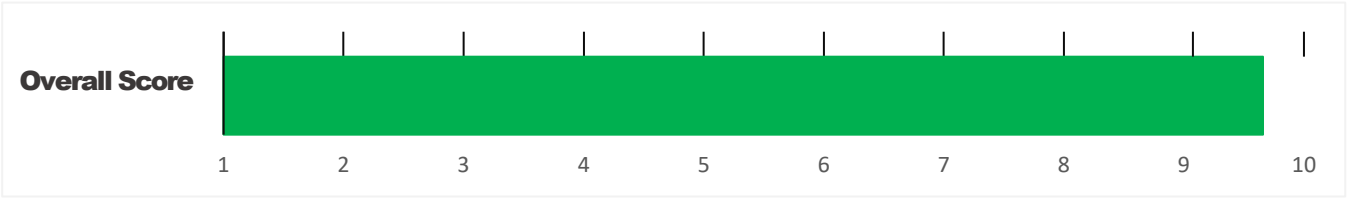
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) 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	1	4	0
Acknowledged	0	0	2	3	11
Resolved	0	0	0	0	0
Noteworty onlyOwner Privileges	Set Taxes and Ratios, Airdrop, Set Protection Settings, Set Reward Properties, Set Reflector Settings, Set Swap Settings, Set Pair and Router				

FENTURE FINANCE Smart contract has achieved the following score: %97.0



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	○ **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.

ℹ 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.**

ℹ️ **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 recommen- dations.

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
TRT-03	Informational	In updateFormapping , the following equation is used inside an unchecked block	Inconsistency	Acknowledged
TDL-04	Inconsistency	In Suggested Constant/Immutable Usages For Gas Efficiency	Coding Practice	Acknowledged
TJL-05	Medium	In Improved Logic of Pool:: addReserveToList()	Business Logic	Acknowledged
TKL-06	Low	UserConfiguration:: getFirstAssetAsCollateralId()	User Configuration	Acknowledged
TCL-07	Informational	Redundant State/Code Removal	Coding Practice	Acknowledged
TXL-08	High	Proper Asset Price in GenericLogic::calculateUserAccountData()	GenericLogic	Acknowledged
TEL-09	Pool	Proper EMode Category Use in Pool::borrow()	Coding Practice	Acknowledged
THL-10	BorrowLogic, UserConfiguration	Possible Underflow Avoidance in BorrowLogic And UserConfiguration	Coding Practice	Acknowledged
TQL-11	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	8oippkjjjk96be0fd50f8632d8433cccc9db6f4b39f9e566d1yhhg8765fffckiuybb
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	4566efgywqutfeuh87872t15378837983639293763hhegetgjfwjk89336668862
TOP	Tivel-contracts-zk/tree/master/contracts	cC51D48	Pool.sol	546363ttebnve88329973mvvdsggct478153ytgdfdxy792635fgdjgi1900990908
TDP	Tivel-contracts-zk/tree/master/contracts	cC51D49	WithdrawalMonitor.sol	835656990327hudbinnjntr6729dchjld0993ytyy3vq63235727879889073
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	455687gfesadjknlppiuhhg774580vgfxrki9876dhgvb990lkjhde444566788
TGV	Tivel-contracts-zk/tree/master/contracts	cC51D85	NonfungiblePositionManager.sol	hbgyyyutwi7653896793jjfohjklfnwwwqafenoggyueppjete38543

AUTOMATED ANALYSIS

Symbol	Definition
🛑	**Function modifies state**
💳	**Function is payable**
🔒	**Function is internal**
🔐	**Function is private**
❗	**Function is important**

TIVEL FINANCE	Interface			
└	totalSupply	External ❗		❗		NO❗
└	decimals	External ❗		❗		NO❗
└	symbol	External ❗		❗		NO❗
└	name	External ❗		❗		NO❗
└	getOwner	External ❗				NO❗
└	balanceOf	External ❗		❗		NO❗
└	transfer	External ❗	"	❗	🛑	|NO❗
└	allowance	External ❗		❗		NO❗
└	approve	External ❗	"	❗	🛑	|NO❗
└	transferFrom	External ❗	"			NO❗

||||||

IFactoryV2	Interface		
└	getPair	External ❗		|NO❗
└	createPair	External ❗	"	|NO❗

||||||

IV2Pair	Interface		
└	factory	External ❗		|NO❗
└	getReserves	External ❗		|NO❗
└	sync	External ❗	"	|NO❗

IRouter01	Interface			
└	factory	External ❗		NO❗
└	SOL	External ❗		NO❗
└	addLiquidityETH	External ❗	#	NO❗
└	addLiquidity	External ❗	"	NO❗
└	swapExacETHForTokens	External ❗	#	NO❗
└	getAmountsOut	External ❗		NO❗
└	getAmountsIn	External ❗		NO❗
IRouter02	Interface	IRouter01		
└	swapExactTokensForAPTSupportingFeeOnTransferTokens	External ❗	"	NO❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	#	NO❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	" ! 🛑	NO❗
└	swapExactTokensForTokens	External ❗	"	NO❗
Protections	Interface			
└	checkUser	External ❗	" ! 🛑	NO❗
└	setLaunch	External ❗	"	NO❗
└	setLpPair	External ❗	"	NO❗
└	ETH	External ❗	"	NO❗
└	removeSniper	External ❗	"	NO❗
Cashier	Interface			
└	setRewardsProperties	External ❗	"	NO❗
└	tally	External ❗	"	NO❗
└	load	External ❗	#	NO❗
└	cashout	External ❗	"	NO❗
└	giveMeWelfarePlease	External ❗	"	NO❗
└	getTotalDistributed	External ❗		NO❗
└	getUserInfo	External ❗		NO❗
└	getUserRealizedRewards	External ❗		NO❗

| └ | getPendingRewards | External ❗ | | |NO❗ |

| └ | initialize | External ❗ | " | |NO❗ |

| └ | getCurrentReward | External ❗ | | |NO❗ |

||||||

| **SOL** | Implementation | SafeMath |||

| └ | <Constructor> | Public ❗ | | # |NO❗ |

| └ | transferOwner | External ❗ | " | | onlyOwner |

| └ | renounceOwnership | External ❗ | " | | NO❗|

| └ | setOperator | Public ❗ | " | |NO❗ |

| └ | renounceOriginalDeployer | External ❗ | " | |NO❗ |

| └ | <Receive ETH> | External ❗ | | # |NO❗ |

| └ | totalSupply | External ❗ | | |NO❗ |

| └ | decimals | External ❗ | | |NO❗ |

| └ | symbol | External ❗ | | |NO❗ |

| └ | name | External ❗ | | |NO❗ |

| └ | getOwner | External ❗ | | ! |NO❗ |

| └ | balanceOf | Public ❗ | | ! |NO❗ |

| └ | allowance | External ❗ | | ! |NO❗ |

| └ | approve | External ❗ | " | ! 🛑 |NO❗ |

| └ | _approve | Internal $ | " 🔒 🛑 | |

| └ | approveContractContingency | Public ❗ | " | ! 🛑 | onlyOwner |

| └ | transfer | External ❗ | " | |NO❗ |

| └ | transferFrom | External ❗ | " | |NO❗ |

| └ | setNewRouter | External ❗ | " | | onlyOwner |

| └ | setLpPair | External ❗ | " | | onlyOwner |

| └ | setInitializers | External ❗ | " | | onlyOwner |

| └ | isExcludedFromFees | External ❗ | | |NO❗ |

| └ | isExcludedFromDividends | External ❗ | | |NO❗ |

| └ | isExcludedFromProtection | External ❗ | | |NO❗ |

| └ | setDividendExcluded | Public ❗ | " | | onlyOwner |

| └ | setExcludedFromFees | Public ❗ | " | | onlyOwner |

TTY-01 Key Findings

Category	Severity ●	Location	Status
Status Mathematical Operations	**Low**	Contract/Pool.sol	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);
            }
        }
```

A transfer call made in this contract may be unstable and cause tokens to become stuck.

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	Low

Description

In **updateForOwner**, Relevant Function Snippet

```
if (_updater != pos.owner && !needLiquidate)
            revert NotOwner(pos.owner, _updater);
        pos.status.isClosed = true;
        pos.closer = _updater;
```

Description

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

Recommendation

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

TRT-03 POSSIBLE OVERFLOW

Category	Severity	Location	Status
Inconsistency	**Informational**	Contract/code/metaaggregator.sol	Low

Description

In **updateFormapping ,** the following equation is used inside an unchecked block

```
contract MetaAggregator is IMetaAggregator, Lockable {
    address public manager;
    mapping(address => bool) acceptedAdapters;
    mapping(address => address) approvalAddress;
;
```

The function mapping () does not have the override specifier. It should be noted that since (> a function that overrides only a single interface function does not require the override specifier (see doc). However, all other instances of this in the code base contain the override specifier.

Recommendation

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

TDL-04 Key Findings

Category	Severity	Location	Status
Coding Practices	Low	Contracts/withdrawalmonitor.sol	Informational

Description

In **Suggested Constant/Immutable Usages For Gas Efficiency**

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

Description

Since version v0.8.10+, 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.

This feature is introduced based on the observation that the reading and writing of storage-based contract states are gas-expensive. Therefore, it is always preferred if we can reduce, if not eliminate, storage reading and writing as much as possible. Those state variables that are written only once are candidates of immutable states under the condition that each fits the pattern, i.e., “a constant, once assigned in the constructor, is read-only during the subsequent operation.”

In the following, we show a number of key state variables defined in PriceOracleSentinel, including _addressesProvider, _oracle, and _gracePeriod. If there is no need to dynamically update these state variables, they can be declared as either constants or immutable for gas efficiency. In particular, the above three states can be defined as immutable.

Recommendation

Revisit the state variable definition and make extensive use of constant/ immutable states.

TJL-05 Key Findings

Category	Severity	Target	Status
Business Logic	**Medium**	**Contract/Pool.sol**	Informational

Description

In **Improved Logic of Pool::_addReserveToList()**

```
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

The Tivel Finance protocol allows the governance to dynamically add new reserves into the protocol. To keep track of the list of active reserves, the protocol maintains the internal state `_reservesList`. While reviewing the accounting of active reserves, we notice the internal routine to add a new reserve needs to be improved.

To elaborate, we show Above the `_BaseReserveToList()` function. It implements a rather straight- forward logic in validating the new asset and then adding it into the internal `_reservesList`. It comes to our attention that the internal `for`-loop needs to terminate the execution once a vacant spot is located and populated. Note the current implementation will simply fill all available slots with the new reserve asset.

Recommendation

Revise the above `_BassReserve()` function to proper add a new reserve asset.

TKL-06 Key Findings

Category	Severity	Target	Status
UserConfiguration	**low**	**(UserConfiguration)** Factory.sol	Acknowledge

Description

UserConfiguration::_getFirstAssetAsCollateralId()

```
    function setCollateralMUT(
        address[] memory _collaterals,
        uint256[] memory _muts
    ) external onlyManager {
        if (_collaterals.length != _muts.length)
            revert BadLengths(_collaterals.length, _muts.length);
        for (uint256 i = 0; i < _collaterals.length; i++) {
            if (_muts[i] > 10000) revert TooHighValue(_muts[i], 10000);
            collateralMUT[_collaterals[i]] = _muts[i];
```

Description

The TIVEL FINANCE protocol has a flexible mechanism to keep track of the configuration of current protocol users. This mechanism is mainly implemented in the `UserConfiguration` contract. In the process of reviewing this contract, we notice an internal helper function can be simplified

To elaborate, we show below this helper routine, i.e., `_getFirstAssetAsCollateralId()`. As the name indicates, this routine is designed to return the address of the first asset used as collateral by the user. It turns out the `collateralData & ~(collateralData - 1)` computation is unnecessary and the step size of 2 can be avoided as well.

Recommendation

Simplify the above routine as the follows:

```
    function setCollateralLT(
        address[] memory _collaterals,
        uint256[] memory _lts
    ) external onlyManager {
        if (_collaterals.length != _lts.length)
            revert BadLengths(_collaterals.length, _lts.length);
        for (uint256 i = 0; i < _collaterals.length; i++) {
            if (_lts[i] > 10000) revert TooHighValue(_lts[i], 10000);
            collateralLT[_collaterals[i]] = _lts[i];
```

UserConfiguration::_getFirstAssetAsCollateralId()

TCL-07 Key Findings

Category	Severity ●	Target	Status
Coding Practices	**Informational**	**Contract/**liquidationmarker.sol	Low

Description

Redundant State/Code Position

```
function performUpkeep(bytes calldata _performData) external override {
        IPositionStorage positionStorage = IPositionStorage(
            positionStorageAddress
        );
        uint256 usedGas = gasleft();
        (bytes32[] memory batchPositionKeys, uint256 count) = abi.decode(
            _performData,
            (bytes32[], uint256)
        );
```

Description

The TIVEL FINANCE protocol makes good use of a number of reference contracts, such as ERC20, SafeERC20, SafeMath, and Address, to facilitate its code implementation and organization. For example, the Pool smart contract has so far imported at least five reference contracts. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

For example, if we examine closely the ReserveLogic library, there is an AccrueToTreasuryLocalVars structure with a number of member fields that are defined, but not used. Examples include the YieldStableRate and stableSupplyUpdatedTimestamp fields. Also, another structure UpdateInterestRatesLocalVars defines an unused member field YieldStableRate.

Recommendation

Consider the removal of the redundant state (or code) with a simplified, consistent implementation.

TXL-08 Key Findings

Category	Severity	Target	Status
Coding Practices	High	Contract/positionstorage.sol	Informational

Description

Proper Asset Price in GenericLogic::calculateUserAccountData()

```
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 Yield Lend protocol is no exception. While reviewing the implementation to measure the debt position, we notice the key function calculateUserAccountData() needs to be improved.

To illustrate, we show below this function. As the name indicates, the function is dedicated to calculate the user data across the reserves. For this end, it requires the total liquidity/collateral/bor- row balances in the base currency used by the price feed, as well as the average loan to value (LVT), the average liquidation ratio, and the health factor. However, it misuses the eModeAssetPrice as the price for each iterated reserve (lines 618-628), which leads to erroneous calculation of collateral value and borrow power. This issue is possibly introduced to support the eMode feature, but has been mistakenly used to consider all reserve assets to be part of the same eMode category.

.

Recommendation

Apply the right price oracle in the above calculateUserAccountData() routine to compute the user account data.

.

TEL-09 Key Findings

Category	Severity ●	Target	Status
Coding Practices	**Medium**	**Contract/**withdrawalmonitor.sol	Informational

Description

Proper EMode Category Use in Pool::borrow()

```
function setFactory(address _factory) external {
        if (factory != address(0)) revert InitializedAlready();
        factory = _factory;
        poolDeployer = IFactory(_factory).poolDeployer();
    }
```

Description

The Tivel Finance protocol has a nice feature `credit delegation`, which allows a credit delegator to delegate the credit of their account's position to a Lender. This feature requires proper accounting of delegation allowance and actual expenditure. While examining its implementation, we notice a key function `borrow()` does not properly follow the `credit delegation` logic.

To elaborate, we show Above this `borrow()` function. This is a core lending function and is used to borrow funds from the lending protocol. It comes to our attention that the encapsulated `DataTypes. ExecuteBorrowParams` parameters mistakenly uses `_usersEModeCategory[msg.sender]` as the user's eMode category. In the `credit delegation` situation, the real eMode category should be `_usersEModeCategory[ onBehalfOf]`.

Recommendation

Ensure the `credit delegation` feature is consistently honored in all aspects of the lending protocol.

THL-10 Key Findings

Category	Severity ●	Target	Status
Coding Practices	**low**	**Contract/**Positionstorage.sol	Informational

Description

Possible Underflow Avoidance in BorrowLogic And UserConfiguration

```
    function getMinCollateralAmount(
        OpenTradePositionParams memory _params
    ) external view override returns (uint256) {
        IFactory _factory = IFactory(factory);
        IPriceFeed priceFeed = IPriceFeed(_factory.priceFeed());
        uint256 pricePrecision = PRICE_PRECISION;

        uint256 baseValue;
        {
```

Description

The TIVEL FINANCE protocol has established itself as one of the leading lending protocol. Within each lending proto-col, 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 0.8.10, 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 isUsingAsCollateralOne() function, it may revert if the current collateralData (line 605) is equal to 0. Another example is when the underlying asset of a reserve has an unusual decimal,

which may revert the following calculation of reserveCache.reserveConfiguration. getDecimals()-ReserveConfiguration.DEBT_CEILING_DECIMALS.

Note this calculation appears in a num- ber of routines. Its revert may bring in unnecessary frictions and cause issues for integration and composability.

Recommendation

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

TQL-11 Key Findings

Category	Severity ●	Target	Status
Coding Practices	**low**	**Contract/**DEXAggregator.sol	**Fixed**

Description

Possible Underflow Avoidance in BorrowLogic And UserConfiguration

```
function getAmountOut(
        address /* _dex */,
        address _tokenIn,
        address _tokenOut,
        uint256 _amountIn
    ) external view override returns (uint256 amountOut, address dex) {
        dex = address(0);
        address[] memory path;
        address bridge = bridgeToken[_tokenIn][_tokenOut];
        if (bridge == address(0)) {
            path = new address[](2);
            path[0] = _tokenIn;
            path[1] = _tokenOut;
        } else {
    ...
```

Description

For gas efficiency, the TIVEL FINANCE protocol is engineered with the reserve cache mechanism, which necessi-tates 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 cer- tain inconsistency in the reserve cache usages and the inconsistency needs to be resolved to avoid confusions and errors.

Recommendation

Revise the above functions to following a consistent approach to 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

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 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 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.

No function to retrieve ownership

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

Owner cant change balance

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

Honeypot Risk

This does not appear to be a honeypot

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

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 Anti Whale

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

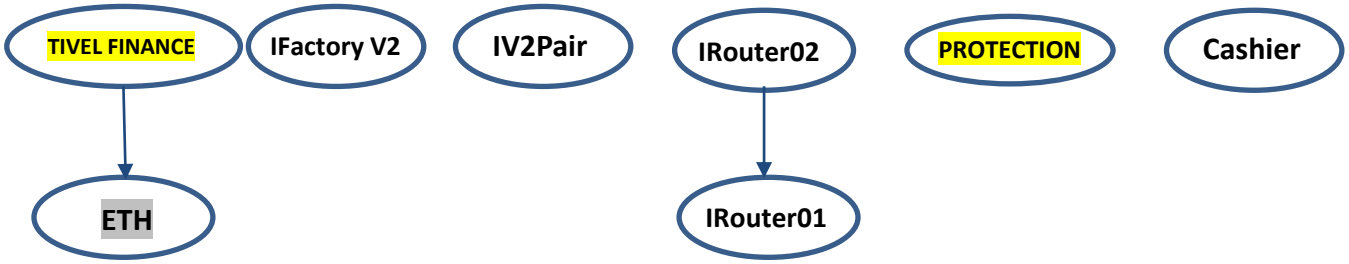
No blacklist function

No blacklist function is included.

No whitelist function

Whitelist function found

INHERITANCE GRAPH



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

Vulnerability 0 : No important security issue detected.
Threat level: Low

```
Home  factory.sol 1  pool.sol 1  pooldeployer.sol 1  witdrawalmonito.sol 1  positionstorage.sol 1  pricefeed.sol 1
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;
0  Listen on all transactions  Filter with transaction hash or address
```

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

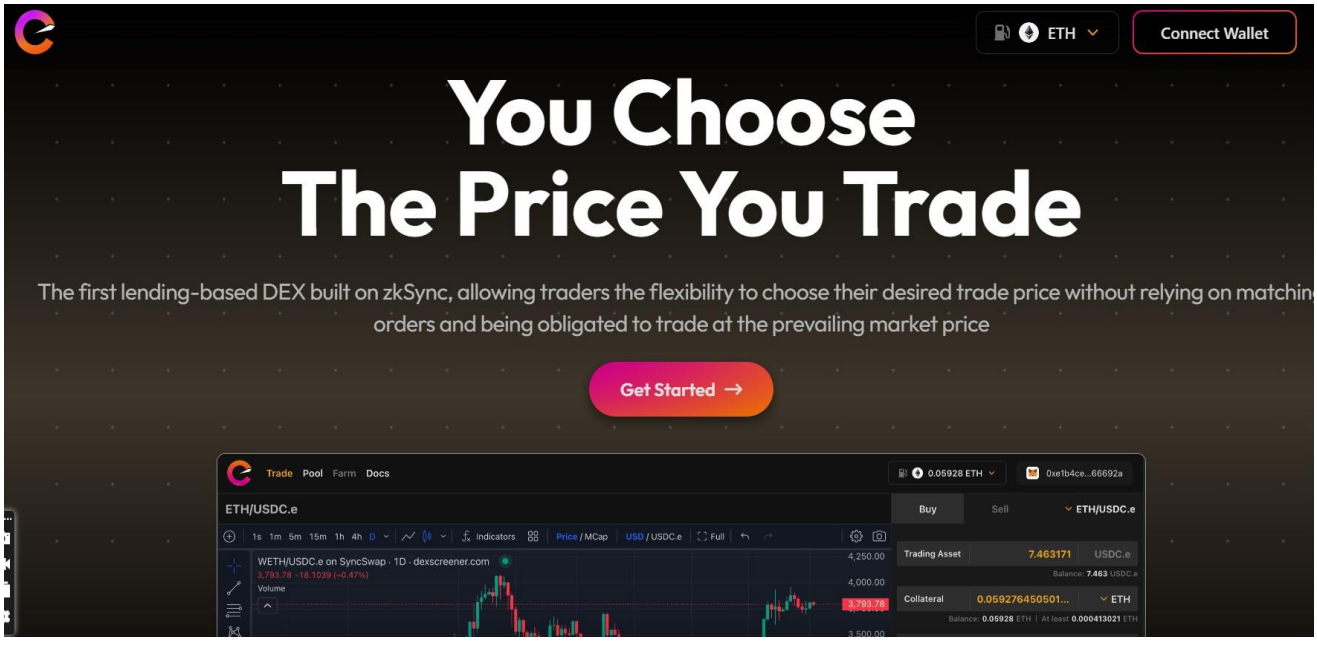
Lending-based DEX is the NEXT GEN of spot trading, allowing traders the flexibility to choose their desired trade price without relying on matching orders and being obligated to trade at the prevailing market price. Traders receive the results of trades instantly from single-token liquidity pools. To 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

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