VITALBlock security.

Blockchain Security | Smart Contract Audit | KYC Certification | SAFU | CEX Listing | Marketing

MADE IN CANADA

TOKEN SALE

AUDIT

SECURITY ASSESSMENT

9th October 2025

For

TOKEN SALE

Making Blockchain, Defi And Web3 A Safer Place.

Smart Check SLITHER TRAIL OF BITS MythX

@VB_Audit @Vitalblock @VB_Audit www.vitalblock.org

CONTENTS

INTRODUCTION

Auditing Firm	**VITAL BLOCK SECURITY**
Client Firm	Token Sale
Methodology	**Automated Analysis, Manual Code Review**
Language	**Solidity**
Contract Code	TokenSale.sol
Source Code Light	**Private Source**
Centralization	**Active ownership**
License	MIT
Dependencies	**OpenZeppelin Contracts (v5+ compatible)**
Solidity Version	**^0.8.28**
Inheritance:	**> ReentrancyGuard** **> AccessControl** **> Pausable** **Uses** **SafeERC20**
Prelim Report Date	**October 9TH 2025**
Final Report Date	**October 9TH 2025**

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

Document Properties

Client	TOKENSALE
Title	Smart Contract Audit Report
Target	TOKENSALE
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	October 9th, 2025	James BK	Final Released
1.0-AP	October 9th , 2025	Jimmy Cole	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.

- TOKENSALE (TN79750)
- https://github.com/ahmetcan-a11y/contr/blob/main/TokenSale%20v2.sol (TN79750)

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*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	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 Security will conduct the smart contract audit of its Sol source code. The audit scope of work is strictly limited to mentioned .SOL file only.

0.Tokensale.sol

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

Verify audited contract code Repo.

Public Contract Code Link:
https://github.com/ahmetcan-a11y/contr/blob/main/TokenSale%20v2.sol

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 Security 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.**

Table 1.0 The Full Audit Checklist

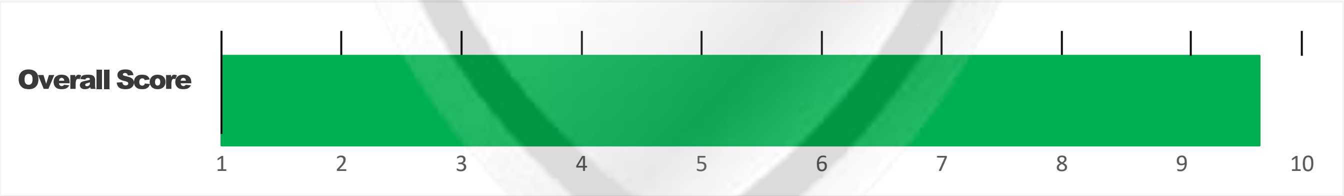
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) 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 **TOKENSALE** 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	0	2
Acknowledged	1	0	2	0	0
Resolved	1	0	1	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				

TOKENSALE Smart contract has achieved the following score: 95.0 %



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.

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 iselevated.

There are some well-intended reasons haveprivileged 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 andmagnitude.

- 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 privilegedroles.
- Remove functions with elevated centralization risk.

ℹ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with arelease schedule.

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 canlead 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 deterexploits.**
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 riskuncertainty.**

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

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

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), 1 High-severith, 2 medium-severity vulnerabilities, 1 low-severity vulnerabilities, and 1 informational recommen- dations.

Table 2.1: Key **TOKENSALE** Audit Findings

ID	Severity	Title	Category	Status
CNY-001	High	ETH Forwarding Vulnerability in receive()/fallback()	Coding Practice	Fixed
CTY-002	Informational	In Inconsistent Error Message in updatePurchaseLimits()	Business Logic	Fixed
CST-003	Low	In Potential Integer Division Truncation in Token Calculation	Status Mathematical Operations	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.

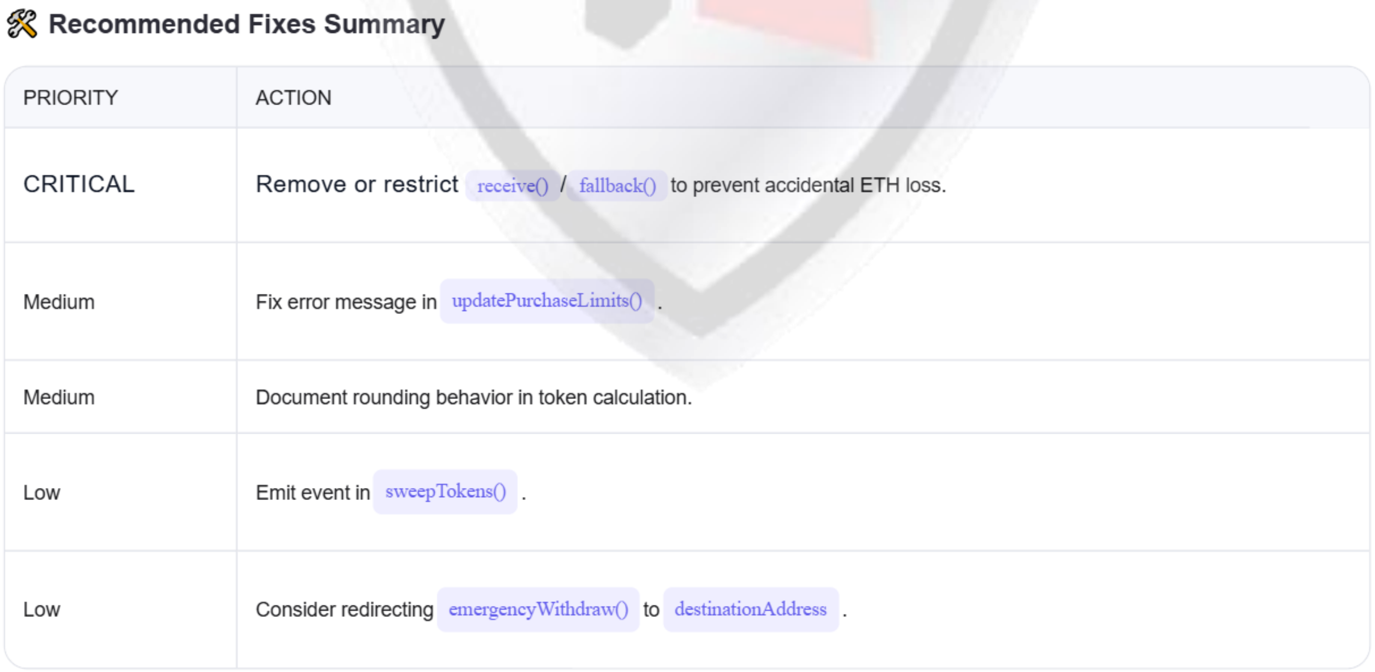
AUTOMATED ANALYSIS

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

```
| ** TOKENSALE ** | 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❗ |
| └ | ETH | External ❗ |  |NO❗ |
| └ | addLiquidityETH | External ❗ |  # |NO❗ |
| └ | addLiquidity | External ❗ | "  |NO❗ |
| └ | swapExacETHForTokens | External ❗ |  # |NO❗ |
| └ | getAmountsOut | External ❗ |  |NO❗ |
| └ | getAmountsIn | External ❗ |  |NO❗ |
||||||
| **IRouter02** | Interface | IRouter01 |||
| └ | swapExactTokensForETHSupportingFeeOnTransferTokens | External ❗ | "  |NO❗ |
| └ | swapExactETHForTokensSupportingFeeOnTransferTokens | External ❗ |  # |NO❗ |
| └ | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ❗ | " ❗ 🛑 |NO❗ |
| └ | swapExactTokensForTokens | External ❗ | "  |NO❗ |
||||||
| **Protections** | Interface |  |||
| └ | checkUser | External ❗ | " ❗ 🛑 |NO❗ |
| └ | setLaunch | External ❗ | " ❗ 🛑 |NO❗ |
| └ | setLpPair | External ❗ | " ❗ 🛑 |NO❗ |
| └| TOKENSALE | 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❗ |
||||||
| **ETH** | Implementation | SafeMath |||
| └ | <Constructor> | Public ❗ |         #  |NO❗ |
| └ | transferOwner | External ❗ | "           | onlyOwner |
| └ | renounceOwnership | External ❗ | "           | NO❗|
| └ | setOperator | Public ❗ | "            |NO❗ |
| └ | renounceOriginalDeployer | External ❗ | "            |NO❗ |
| └ | <Receive Ether> | 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 |
```

OPTIMIZATIONS | TOKENSALE

ID	Title	Category	Status
CTV	**Logarithm Refinement Optimization**	Gas Optimization	Acknowledged ●
COP	**Checks Can Be Performed Earlier**	Gas Optimization	Acknowledged ●
CDP	**Unnecessary Use Of SafeMath**	Gas Optimization	Acknowledged ●
CWY	**Struct Optimization**	Gas Optimization	Acknowledged ●
CGT	**Unused State Variable**	Gas Optimization	Acknowledged ●



🛠 Recommended Fixes Summary

PRIORITY	ACTION
CRITICAL	Remove or restrict `receive()` / `fallback()` to prevent accidental ETH loss.
Medium	Fix error message in `updatePurchaseLimits()` .
Medium	Document rounding behavior in token calculation.
Low	Emit event in `sweepTokens()` .
Low	Consider redirecting `emergencyWithdraw()` to `destinationAddress` .

General Detectors

Transfer Limit

The max/min amount of token transferred can be limited

⚠ Attention Required

DoS with Failed Call

This contract uses external calls that may fail, resulting in loss of functionality

⚠ Attention Required

Division Before Multiplication

The order of operations used may result in a loss of precision.

⚠ 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

TN-01 Key Findings

Category	Severity ●	Target	Status
Business Logic	HIGH	receive() and fallback() functions	Fixed ●

Description

In **update ETH Forwarding Vulnerability in receive()/fallback()**, Relevant Function Snippet

Issue:
The contract uses low-level .call{value: ...}("") to forward ETH. While this avoids reentrancy (due to nonReentrant not applying here), it lacks validation that the destination is a payable contract or EOA. More critically:

> If destinationAddress is a contract without a payable fallback, the ETH transfer reverts, causing the entire transaction to fail.

>However, if the destination accepts ETH but later becomes malicious, it could trap funds.

>Worse: There is no way to recover ETH if destinationAddress becomes invalid (e.g., self-destructed).

But the real critical risk is this:
| "The contract accepts ETH even though the sale is USDT-only.

The presence of receive() and fallback() implies ETH can be sent, but:
>purchaseTokens() only accepts USDT.
>ETH sent to the contract is forwarded blindly, with no accounting, no tokens issued, and no event emitted.
>This creates a user trap: a user might accidentally send ETH expecting tokens and lose funds permanently.
Impact: High – Users can lose ETH with no recourse.

Recommendation:

Remove receive() and fallback() unless ETH payments are explicitly supported.
If ETH support is intended, implement a parallel purchaseWithETH() function with proper token issuance and rate logic.
Otherwise, explicitly reject ETH:.

```
receive() external payable {
    revert("ETH not accepted");
}
```

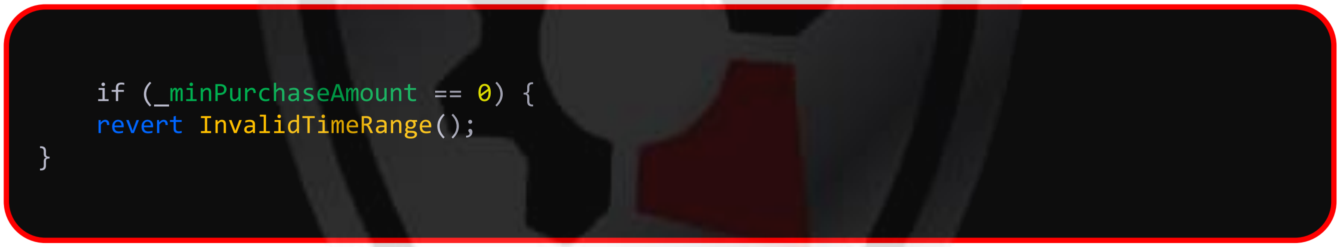
TN-02 Key Findings

Category	Severity ●	Location	Status
Status Mathematical Operations	Medium	updatePurchaseLimits()	Informational

Description

In **Inconsistent Error Message in** updatePurchaseLimits**()**

Issue:



```
    if (_minPurchaseAmount == 0) {
    revert InvalidTimeRange();
}
```

•This reuses InvalidTimeRange() for a non-time-related validation, which is misleading for debugging and monitoring.

Recommendation

Introduce a new error, e.g., MinPurchaseCannotBeZero(), or reuse ZeroAmount().

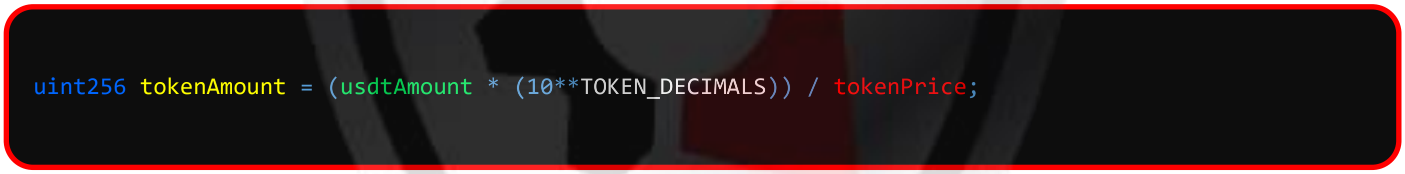
AN-03 POSSIBLE OVERFLOW

Category	Severity ●	Location	Status
Status Mathematical Operations	Medium	purchaseTokens() and calculateTokenAmount()	Acknowledged

Description

In **Potential Integer** Division Truncation in Token Calculation

Issue:



```
uint256 tokenAmount = (usdtAmount * (10**TOKEN_DECIMALS)) / tokenPrice;
```

Since tokenPrice is in USDT decimals (6), but the numerator scales to 18 decimals, the division may truncate small amounts, leading to users receiving fewer tokens than expected (though not a security flaw, it's a fairness/user experience issue).

•Note: This is mathematically correct given the comment "0.2 USDT = 1 Token" → tokenPrice = 200_000 (0.2 * 1e6).

Recommendation

Add a comment clarifying rounding behavior, or consider using a library like FixedPointMath for precise division if fractional tokens matter.
Not critical, but worth documenting.

Vulnerability Scan

REENTRANCY

✓ No reentrancy risk found

Severity	Major
Confidence Parameter	Certain

Vulnerability Description

☒ **Additional Observations:** More amount of the TOKENSALE can **NOT** be minted by a private wallet or contract. **(This is Essentially normal for most contracts)**

Scanning Line:

🟡 Low: emergencyWithdraw() Bypasses Destination Logic

•Location: emergencyWithdraw()

•Issue:
This function sends tokens to msg.sender (admin), not to destinationAddress. While intended for emergencies, it:

- Breaks the invariant that all funds go to destinationAddress.
- Could be misused if admin key is compromised.

•Recommendation:
Consider whether emergency withdrawals should go to destinationAddress instead, or document this as an intentional admin privilege.

🟡 Low: Missing Event in sweepTokens()

•Issue: sweepTokens() does not emit an event, making it hard to track off-chain.

•Recommendation: Emit an event like TokensSwept(address token, uint256 amount).

🟡 Low: Redundant if (msg.value > 0) Check

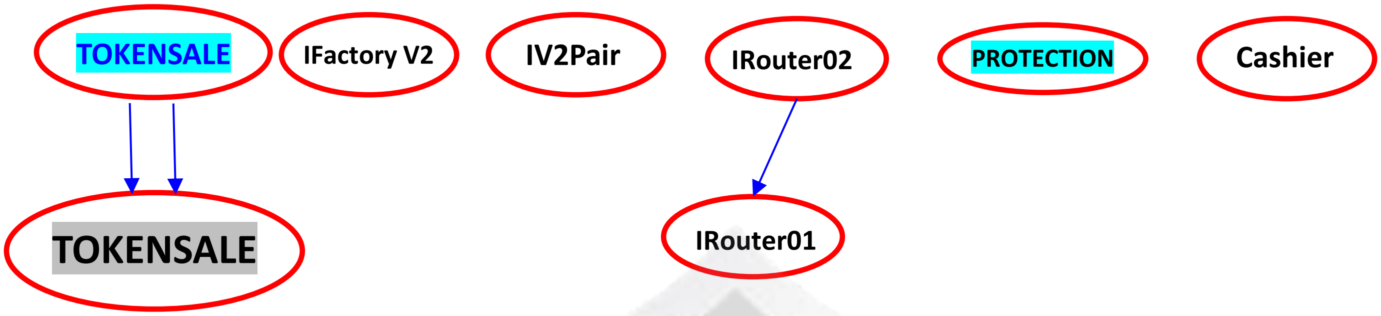
•In receive()/fallback(), msg.value is always > 0 by definition. The check is unnecessary.

Repository:

https://github.com/ahmetcan-a11y/contr/blob/main/TokenSale%20v2.sol

Audited Files	O.Tokensale.sol
Contract Creator Address	Not Established
Deployed Contracts:	Not Deployed
Creator TXH Contracts:	***Not Refillable***

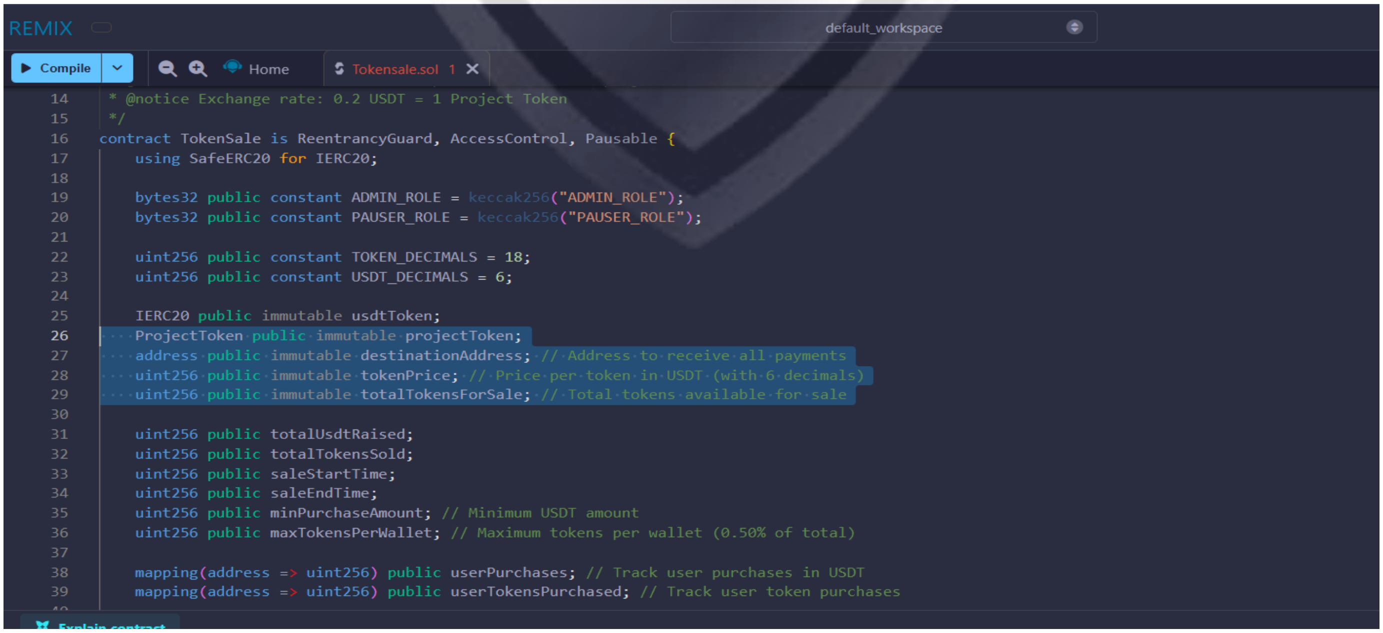
INHERITANCE GRAPH



Identifier	Definition	Severity
CEN-12	Centralization privileges of TOKENSALE	Medium #🟡

Vulnerability 0 : No important security issue detected.

Threat level: Low



```
 * @notice Exchange rate: 0.2 USDT = 1 Project Token
 */
contract TokenSale is ReentrancyGuard, AccessControl, Pausable {
    using SafeERC20 for IERC20;

    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
    bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");

    uint256 public constant TOKEN_DECIMALS = 18;
    uint256 public constant USDT_DECIMALS = 6;

    IERC20 public immutable usdtToken;
    ProjectToken public immutable projectToken;
    address public immutable destinationAddress; // Address to receive all payments
    uint256 public immutable tokenPrice; // Price per token in USDT (with 6 decimals)
    uint256 public immutable totalTokensForSale; // Total tokens available for sale

    uint256 public totalUsdtRaised;
    uint256 public totalTokensSold;
    uint256 public saleStartTime;
    uint256 public saleEndTime;
    uint256 public minPurchaseAmount; // Minimum USDT amount
    uint256 public maxTokensPerWallet; // Maximum tokens per wallet (0.50% of total)

    mapping(address => uint256) public userPurchases; // Track user purchases in USDT
    mapping(address => uint256) public userTokensPurchased; // Track user token purchases
```

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.	Sol 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 solidity 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 solidity 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 is Normal for most deployer and/or contract owner .

Additional Observations

✅ ProjectToken Assumptions

The audit assumes ProjectToken:

•Is an ERC20 with mint(address, uint256).

•Has a paused() view function.

•Exposes MAX_SUPPLY() as a public constant or view.

•Recommendation: Ensure ProjectToken.mint() is only callable by TokenSale (via onlyOwner or access control).

✅ Role Management

•DEFAULT_ADMIN_ROLE, ADMIN_ROLE, and PAUSER_ROLE are all granted to deployer — acceptable for centralized sales.

•Consider whether PAUSER_ROLE should be separate from ADMIN_ROLE for operational security.

✅ Immutables

•Critical addresses and parameters are immutable — excellent for trust minimization.

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-7 CENTRALIZED PRIVILEGES for a detailed understanding.

ALLEVIATION

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

References

1 MITRE. CWE-1041: Use of Redundant Code. https://cwe.mitre.org/data/definitions/1041.html.

2 MITRE. CWE-1099: Inconsistent Naming Conventions for Identifiers. https://cwe.mitre.org/data/definitions/1099.html.

3 MITRE. CWE-561: Dead Code. https://cwe.mitre.org/data/definitions/561.html.

4 MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

5 MITRE. CWE-663: Use of a Non-reentrant Function in a Concurrent Context. https://cwe.mitre.org/data/definitions/663.html.

6 MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

7 MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

8 MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

9 MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

10 MITRE. CWE CATEGORY: Concurrency. https://cwe.mitre.org/data/definitions/557.html.

11 MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

12 OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

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 Security 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 OTHERWEBSITES

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

VITALBLOCK SMART CONTRACT AUDIT

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

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

Blockchain Security | Smart Contract Audit | KYC Certification | SAFU .

MADE IN CANADA

@VB_Audit Vitalblock.org @Vitalblock