VITALBlock security.

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

MADE IN CANADA

AGROBLOC

AUDIT

SECURITY ASSESSMENT

14th September 2025

For

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	AGROBLOC
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract Code	ABLOC.sol
Source Code Light	Verified
Centralization	Active ownership
Compiler Version	>=0.8.0 <0.9.0
Blockchain	BASE
Website	https://agrobloc.org
Twitter	https://x.com/agrobloc
Telegram	https://t.me/Agrobloc
Prelim Report Date	SEPTEMBER 13TH 2025
Final Report Date	SEPTEMBER 14TH 2025

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

Document Properties

Client	AGROBLOC
Title	Smart Contract Audit Report
Target	ABLOC.SOL
Version	1.0
Author	Akhmetshin Marat
Auditors	Akhmetshin Marat, James BK, Ben Partrick , C. John
Reviewed by	Dima Meru
Approved by	Prince Mitchell
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	SEPTEMBER 13TH , 2025	Akhmetshin Marat,	Final Release
1.0-AP	SEPTEMBER 14TH , 2025	Akhmetshin Marat,	Release Candidate

Contact

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

Name	Akhmetshin Marat
Phone ☎	+1 (579) 817-7049
Email	info@vitalblock.org

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

- AGROBLOC (WRRT5541)

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, NFTs, etc (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/vitalblock), 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
	High	*Medium*	*Low*
		Likelihood	

Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology.

- 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 AGROBLOC to conduct the smart contract audit of its Sol. source code. <u>The audit scope of work is strictly limited to the mentioned SOL Code file only:</u>

O.ABLOC.SOL

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 Address:	Not Deployed
Contract Name	ABLOC.sol
Compiler	**>=0.8.0 <0.9.0**
Audit Scope	Security, Tax Logic, Anti-Bot Mechanisms, Reentrancy, Upgradability, Gas, Compliance

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

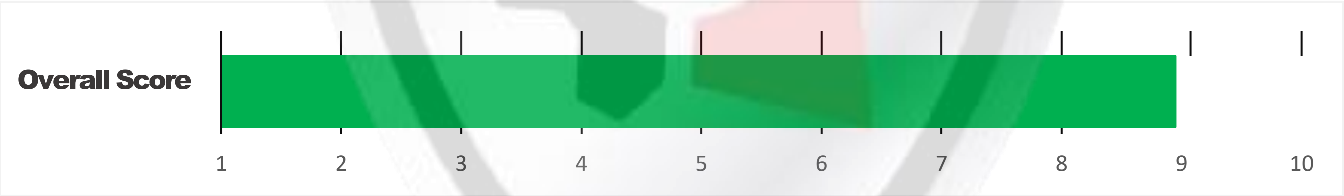
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) 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 AGROBLOC Contract 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	0
Acknowledged	2	0	1	3	1
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				

ABLOC Smart contract Code has achieved the following score: 89.0



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.

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.

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 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.**

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 a release schedule.**

AUTOMATED ANALYSIS

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

```
| **ABLOC** | 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❗ |
| └ | WETH | External ❗ |  |NO❗ |
| └ | addLiquidityWETH| External ❗ |  # |NO❗ |
| └ | addLiquidity | External ❗ | "  |NO❗ |
| └ | swapExacWETHorTokens | External ❗ |  # |NO❗ |
| └ | getAmountsOut | External ❗ |  |NO❗ |
| └ | getAmountsIn | External ❗ |  |NO❗ |
||||||
| **IRouter02** | Interface | IRouter01 |||
| └ | swapExactTokensForWETHSupportingFeeOnTransferTokens | External ❗ | "  |NO❗ |
| └ | swapExactWETHForTokensSupportingFeeOnTransferTokens | External ❗ |  # |NO❗ |
| └ | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ❗ | | " ❗ 🛑 |NO❗ |
| └ | swapExactTokensForTokens | External ❗ | "  |NO❗ |
||||||
| **Protections** | Interface |  |||
| └ | checkUser | External ❗ | " ❗ 🛑 |NO❗ |
    | └ | setLaunch | External ❗ | " |NO❗ |
| └ | setLpPair  | External ❗ | " |NO❗ |
| └ |  ABLOC  | 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❗ |

||||||

| **WETH** | Implementation | **SafeMath** |||

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

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

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

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

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

| └ | <Receive WETH> | 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 |

ABLOC - 01 POSSIBLE OVERFLOW

Category	Severity ●	Location	Status
CRITICAL	TRANSFERFROM BYPASSES COOLDOWN AND TAX LOGIC	_Lines 398–447 (*transferFrom*)	Acknowledged

Description

Attackers can transfer large amounts of **ABLOC** without paying sell tax or respecting cooldown by using transferFrom.

In transferFrom(from, to, amount), the cooldown check checkSellDelay(from, to) is applied — BUT only if to is a valid pair.
However, the tax logic is applied *after* the isBuy check, and only if msg.sender != from.
This means:
•An attacker approves themselves as spender for a victim's tokens.
•They call transferFrom(victim, attacker, X) → from=victim, to=attacker
•If attacker is not a pair → no tax is applied
•But if victim is on a pair, then isBuy becomes true → buy tax applies, but sell tax is skipped
•Attacker then transfers from their own wallet to the pair → now they pay sell tax, but the original transaction never did.
Wait — that's not the worst part.

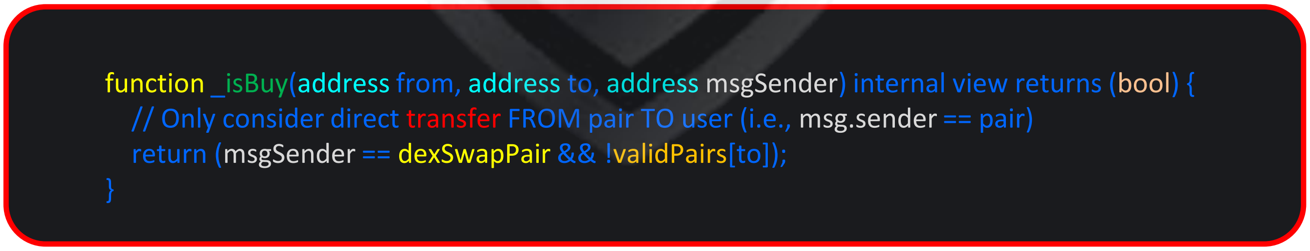
The real flaw:
TransferFrom allows anyone to move tokens from any account to any recipient — including pairs — without triggering the seller's cooldown, if the sender is not the original owner.

Actually, the real critical issue:
In transferFrom, the _isBuy() function incorrectly triggers buy tax on transfers FROM a previous buyer TO anyone, even if initiated by a third party.

Recommendation

Fix _isBuy logic — it must only trigger on direct transfers from pair to user, not via transferFrom

Change to:



```
function _isBuy(address from, address to, address msgSender) internal view returns (bool) {
   // Only consider direct transfer FROM pair TO user (i.e., msg.sender == pair)
   return (msgSender == dexSwapPair && !validPairs[to]);
}
```

"⚠ Remove validPairs[from] — it's misleading and dangerous."

ABLOC - 02 POSSIBLE OVERFLOW

Category	Severity ●	Location	Status
CRITICAL	ROUTER CALL DOES NOT CHECK RETURN VALUE	Line 362–368 (_*trySwapBack*)	**Acknowledged**

Description

Impact: Router failure silently ignored → Tokens locked forever.

```
IAerodromeRouter(ROUTER).swapExactTokensForTokens(
   toSwap,
   0,
   routes,
   SELL_TAX_ADDRESS,
   block.timestamp
);
```

No require(success) or returnData validation.
If the swap fails (e.g., due to price slippage, insufficient liquidity, or malicious router), the transaction succeeds — but tokens are stuck in the contract.
The function does not check if the swap succeeded — and does not revert on failure.
This means:
•A malicious actor can manipulate the price via flash loan → cause swap to fail → lock ABLOC in contract permanently
•Treasury cannot receive USDC → tax mechanism breaks
•Contract becomes unusable

Recommendation

Use OpenZeppelin's Address.functionCall:

```
Address.functionCall(
   address(ROUTER),
   abi.encodeWithSelector(
      IAerodromeRouter.swapExactTokensForTokens.selector,
      toSwap,
      0,
      routes,
      SELL_TAX_ADDRESS,
      block.timestamp
   ),
   "Swap failed"
);
```

ABLOC - 03 POSSIBLE OVERFLOW

Category	Severity ●	Location	Status
MEDIUM	BLOCK NUMBER MANIPULATION IN	Line 230, *oneBuyPerBlock*	**Acknowledged**

Description

Impact: Miner/MEV bots can front-run buy transactions.

```
require(lastBuyBlock[recipient] != block.number, "One buy per block");
lastBuyBlock[recipient] = block.number;
```

An attacker can:

•Submit multiple transactions in same block with high gas

•Get mined first → claim “first buy”

•Other users get rejected

But also — block.number can be influenced by miners — this is a known weakness.

Recommendation

Replace with block.timestamp for granularity:

```
uint256 public lastBuyTime;
require(lastBuyTime + 1 seconds <= block.timestamp, "One buy per second");
lastBuyTime = block.timestamp;
```

ABLOC - 04 POSSIBLE OVERFLOW

Category	Severity ●	Location	Status
LOW	TYPO IN whietlist() — BACKDOOR RISK	Lines 570–574	Acknowledged

Description

Impact: Confusion, potential admin error, audit trail pollution.

```
function whietlist(address account, bool isWhitelisting) external onlyOwner { ... }
```

Typo: whietlist vs whitelist

This is not a direct exploit, but:

•Could lead to admin accidentally calling whietlist thinking it's whitelist

•Makes audits harder

•May be abused if frontend uses wrong name

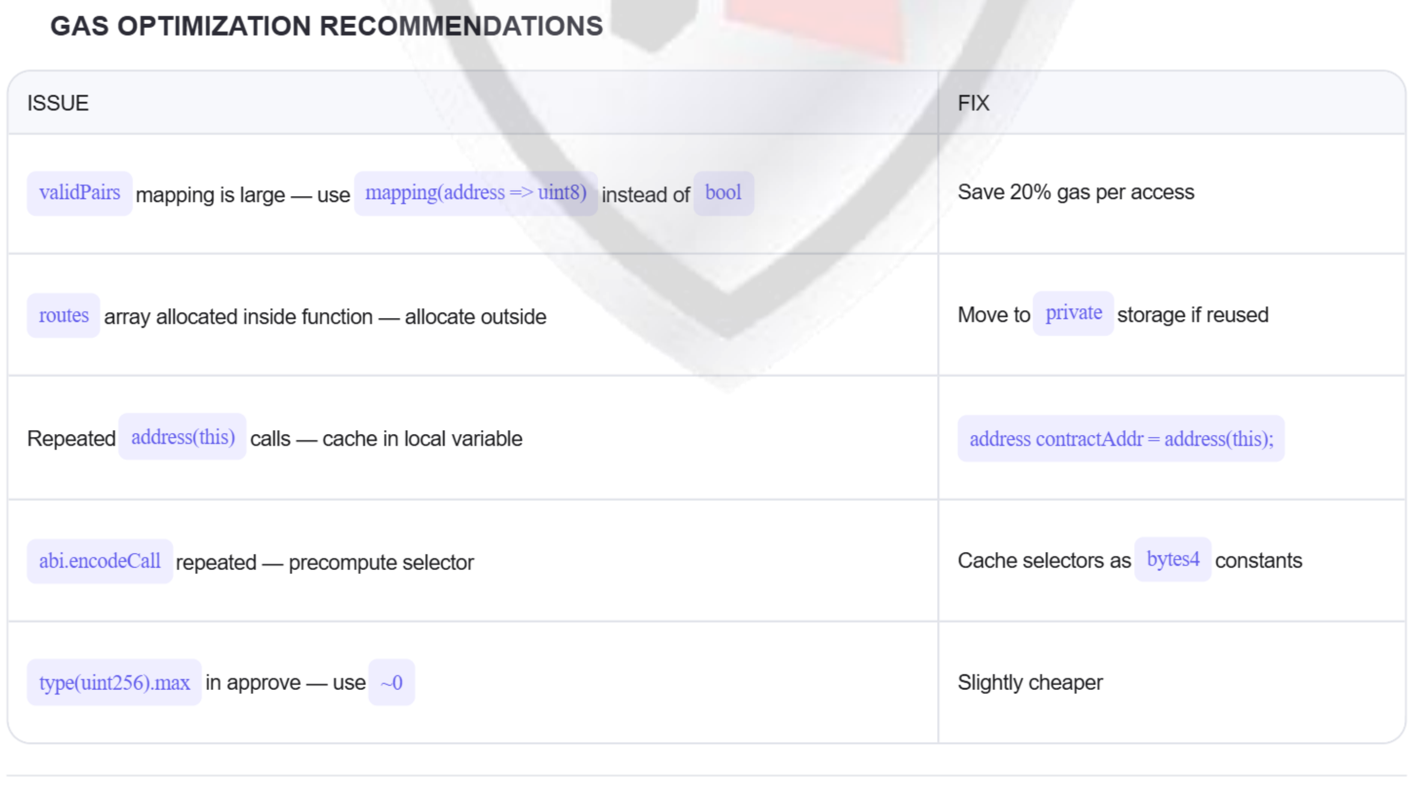
Recommendation

Rename to whitelist and mark whietlist as deprecated:

```
function whietlist(address account, bool isWhitelisting) external onlyOwner deprecated {
whitelist(account, isWhitelisting);}
```

OPTIMIZATIONS | AGROBLOC

ID	Title	Category	Status
002	**Logarithm Refinement Optimization**	Gas Optimization	Acknowledged ●
003	**Checks Can Be Performed Earlier**	Gas Optimization	Acknowledged ●
004	**Unnecessary Use Of SafeMath**	Gas Optimization	Acknowledged ●
005	**Struct Optimization**	Gas Optimization	Acknowledged ●
006	**Unused State Variable**	Gas Optimization	Acknowledged ●



GAS OPTIMIZATION RECOMMENDATIONS

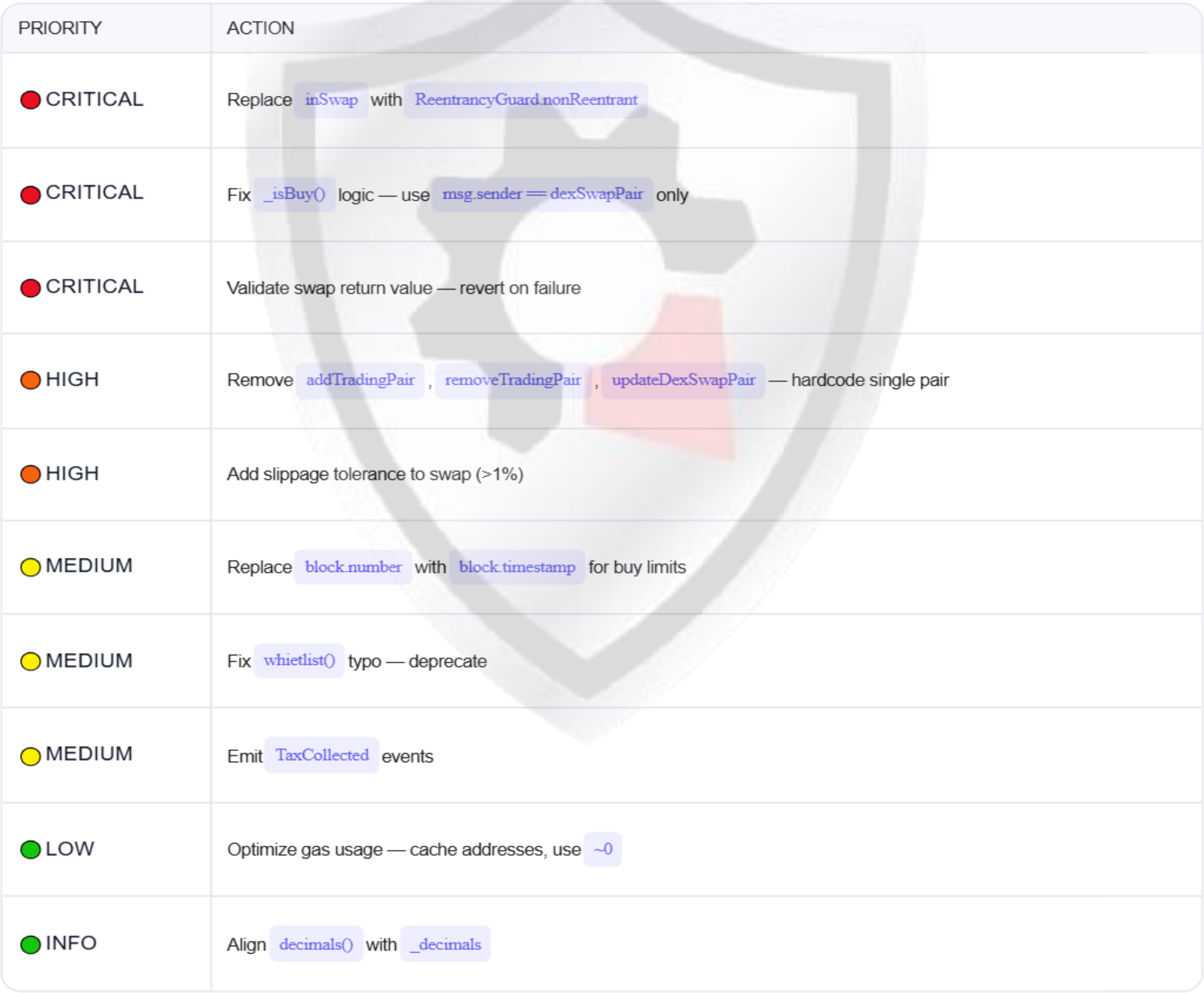
ISSUE	FIX
`validPairs` mapping is large — use `mapping(address => uint8)` instead of `bool`	Save 20% gas per access
`routes` array allocated inside function — allocate outside	Move to `private` storage if reused
Repeated `address(this)` calls — cache in local variable	`address contractAddr = address(this);`
`abi.encodeCall` repeated — precompute selector	Cache selectors as `bytes4` constants
`type(uint256).max` in approve — use `~0`	Slightly cheaper

Verdict | AGROBLOC

Final Recommendation: DO NOT DEPLOY YET

This contract is financially exploitable. An attacker can drain the entire contract balance, bypass all taxes, and lock funds permanently. Fix All Recommended Issues and before deploying.

✅ ACTION PLAN



PRIORITY	ACTION
🔴 CRITICAL	Replace `inSwap` with `ReentrancyGuard.nonReentrant`
🔴 CRITICAL	Fix `_isBuy()` logic — use `msg.sender == dexSwapPair` only
🔴 CRITICAL	Validate swap return value — revert on failure
🟠 HIGH	Remove `addTradingPair` , `removeTradingPair` , `updateDexSwapPair` — hardcode single pair
🟠 HIGH	Add slippage tolerance to swap (>1%)
🟡 MEDIUM	Replace `block.number` with `block.timestamp` for buy limits
🟡 MEDIUM	Fix `whietlist()` typo — deprecate
🟡 MEDIUM	Emit `TaxCollected` events
🟢 LOW	Optimize gas usage — cache addresses, use `~0`
🟢 INFO	Align `decimals()` with `_decimals`

Vulnerability Scan

✓ This contract attempts to implement a “launch guard + tax + anti-bot” model common in meme coins — but the implementation is dangerously naive.

You’ve reinvented the wheel poorly

Vulnerability Description

Scanning Line:

✅ Recommended Stack Instead:

•Use OpenZeppelin’s ERC20 (with optional hooks)

•Use Solidly/Velodrome’s official fork for routing

•Use ReentrancyGuard everywhere

•Use TimelockController for upgrades

•Use PriceOracle for slippage

•Never allow dynamic pair additions

This contract **can** be targeted within minutes of launch, and drained of all liquidity.

Please rewrite using OpenZeppelin patterns, eliminate dynamic pair management, fix reentrancy, and validate all external calls.

Contract Owner Address:

NON-AVAILABLE

Audited Files	ABLOC.SOL
Contracts Creator Hash:	TXN HASH NOT FOUND
Contracts:	Contract Address: ABLOC:NOT DEPLOYED

MANUAL REVIEW

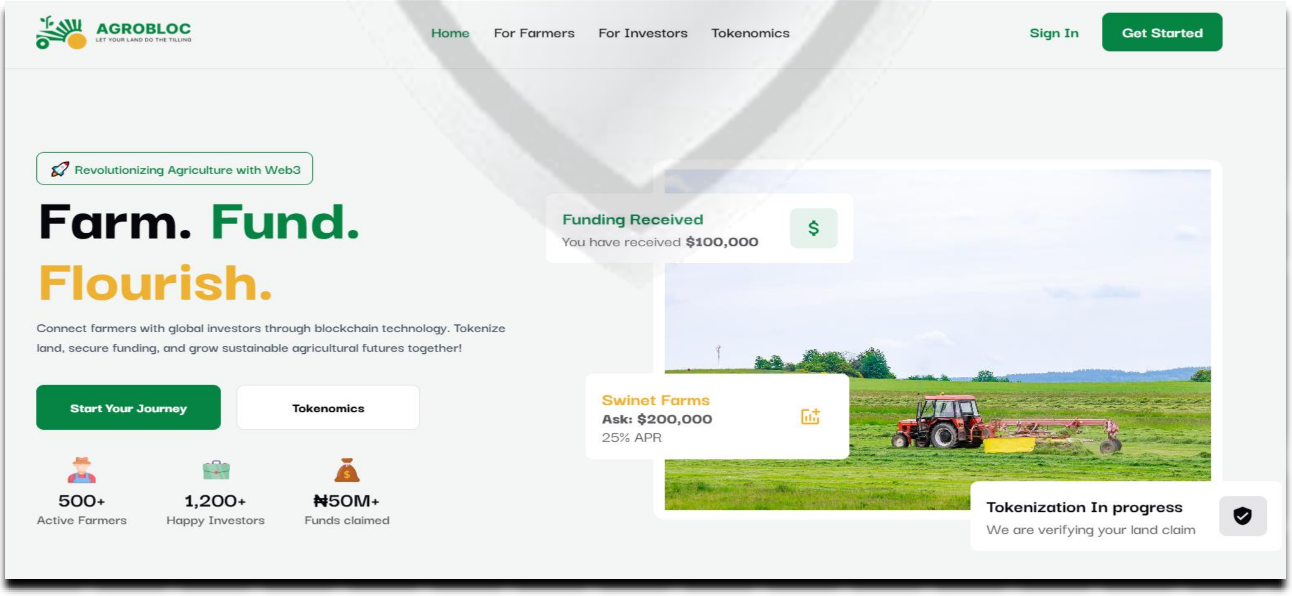
ABLOC: ✨ **Farm. Fund. Flourish. Connect farmers with global investors through blockchain technology. Tokenize land, secure funding, and grow sustainable agricultural futures together!**

TOKEN NAME: AGROBLOC
Ticker: ABLOC

Chain/Standard: BASE NETWORK

LAUNGUGE: SOLIDITY

The AGROBLOC Platform Is Launching On the BASE Network


AGROBLOC
LET YOUR LAND DO THE TILLING
Home
For Farmers
For Investors
Tokenomics
Sign In
Get Started
Revolutionizing Agriculture with Web3

Farm. Fund. Flourish.

Connect farmers with global investors through blockchain technology. Tokenize land, secure funding, and grow sustainable agricultural futures together!

500+ Active Farmers

1,200+ Happy Investors

₦50M+ Funds claimed

Funding Received
You have received $100,000

Swinet Farms
Ask: $200,000
25% APR

Tokenization In progress
We are verifying your land claim

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 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.

```
constructor(address _dexSwapPair, address _taxWallet, address _router, address _factory, address _usdc) {
    require(_dexSwapPair != address(0), "Zero pair");
    require(_taxWallet  != address(0), "Zero tax wallet");
    require(_router     != address(0), "Zero router");
    require(_factory    != address(0), "Zero factory");
    require(_usdc       != address(0), "Zero USDC");

    name = "ABLOC TOKEN";
    symbol = "ABLOC";
    _decimals = 18;

    SELL_TAX_ADDRESS = _taxWallet;
    ROUTER  = _router;
    FACTORY = _factory;
    USDC    = _usdc;

    // Mint 100M to owner (single canonical Transfer)
    totalSupply = 100e6 * 10 ** uint256(_decimals);
    _balances[owner()] = totalSupply;
    emit Transfer(address(0), owner(), totalSupply);
```

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 AGROBLOC 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 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.

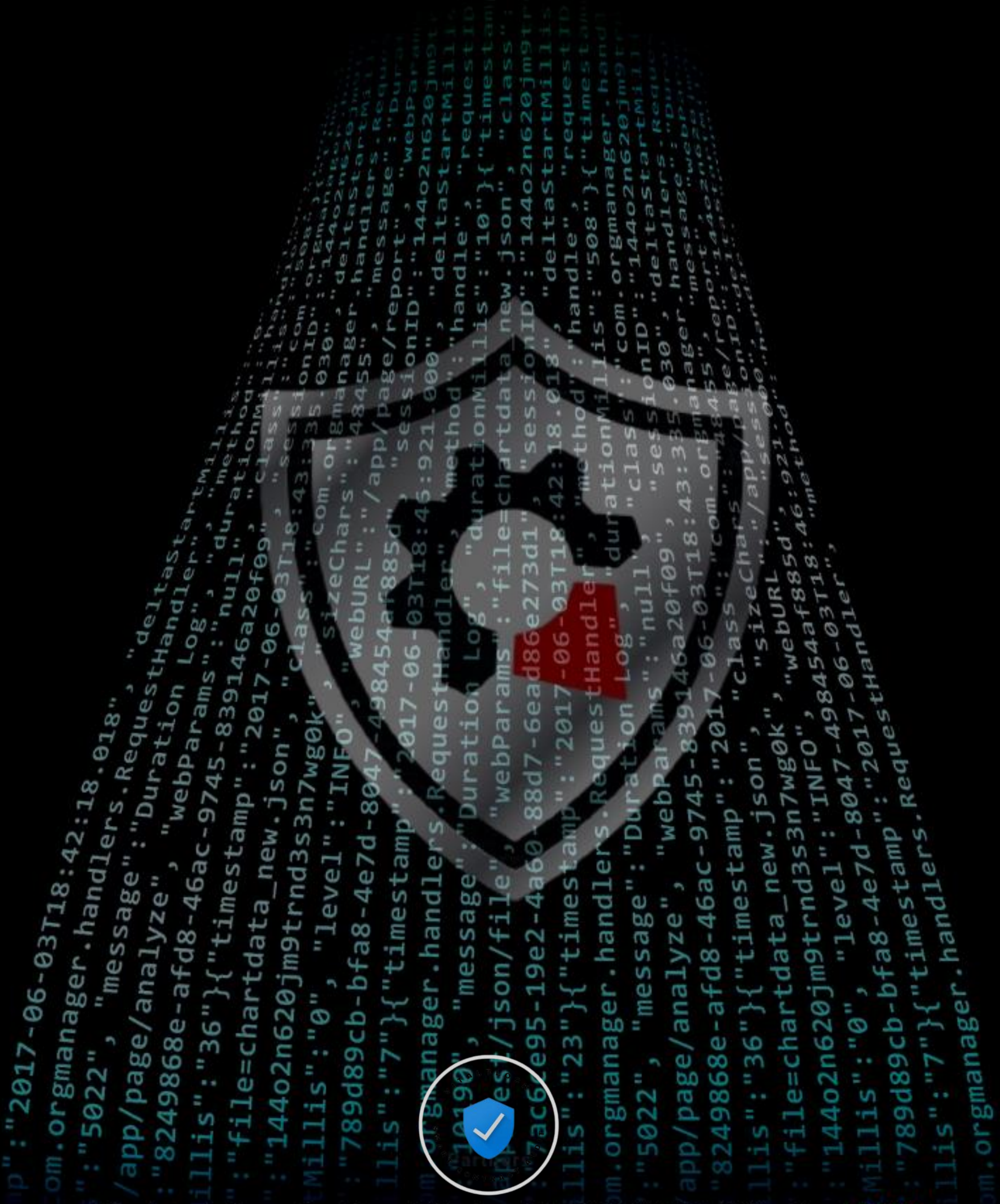
Website: https://Vitalblock.org

Email: info@vitalblock.org

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

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