Blaize.Security

June 15th, 2022 / V. 1.0

VIEWPOINT LABS

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Viewpoint Labs contract's source code was taken from the repository provided by the Viewpoint Labs team.

SCORE **9.1** /10

The scope of the project is **Viewpoint Labs** set of contracts**:**

1/ KaleToken.sol

2/ TokenDistributor1.sol

3/ TokenDistributor2.sol

Repository:

https://github.com/viewpoint-labs/smart-contracts

Primary SHA256 (audited):

- 3649fd10e4cf7a29d189fabcdacb6e365de0a568

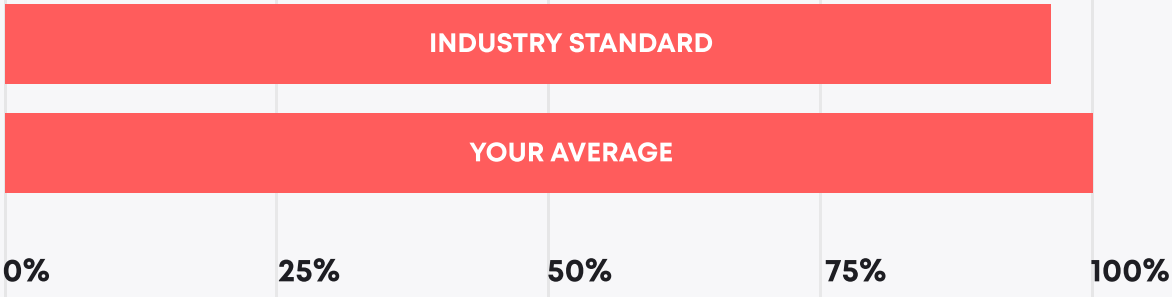
Final SHA256 (post-audit):

- 88b34ba72b278f956120d3d306f6a2ff2f6db9ae

TECHNICAL SUMMARY

In this report, we consider the security of the contracts for Viewpoint Labs protocol. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of **Viewpoint Labs** smart contracts conducted between **May 24th, 2022 - June 15th, 2022.**

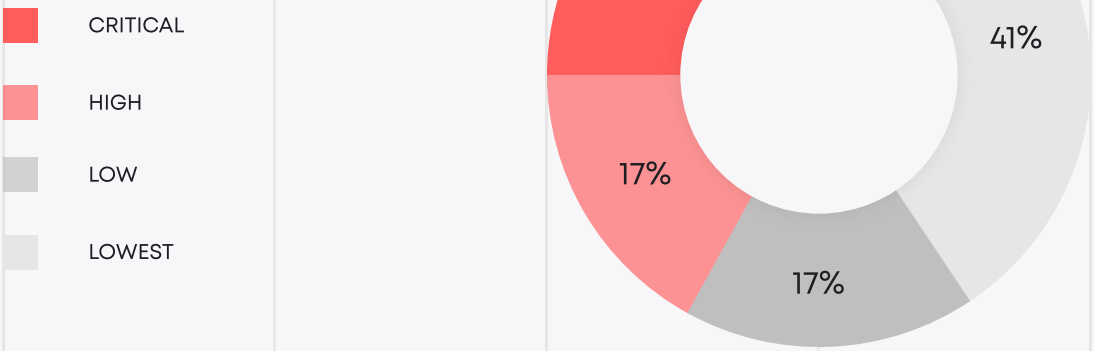
Testable code



The testable code is 100%, which corresponds to the industry standard of 95%.

The scope of the audit includes the unit test coverage, that bases on the smart contracts code, documentation and requirements presented by the Viewpoint Labs team. Coverage is calculated based on the set of Truffle framework tests and scripts from additional testing strategies. Though, in order to ensure a security of the contract Blaize.Security team recommends the Viewpoint Labs team put in place a bug bounty program to encourage further and active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of found issues and their severity. A total of 12 problems were found. 12 issues were fixed or verified by the Viewpoint Labs team.

	FOUND	FIXED/VERIFIED
Critical	3	3
High	2	2
Medium	0	0
Low	2	2
Lowest	5	5

SEVERITY DEFINITION

Critical

A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.

High

A system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge information or financial leak. Needs immediate improvements and further checking.

Medium

A system contains issues which may lead to mediumfinancial loss or users' private information leak. Needs immediate improvements and further checking.

Low

A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.

Lowest

A system does not contain any issue critical to the secure work of the system, yet is relevant for best

AUDITING STRATEGYAND TECHNIQUES APPLIED \ PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

Automated analysis:

Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. Checking smart contract logic and comparing it with the one described in the documentation.

EXECUTIVE SUMMARY

Audited protocol represents connected set of contracts: protocol ERC20 token (with fixed supply of 10M tokens), primary tokens distributor for the whole supply (for 6 different wallets on monthly schedule), and the secondary tokens distributor.

The contract contained several critical issues connected to the incorrect token amounts, incorrect distribution schedule, and for tokens deadlocks. Though, the team has fixed issues and corrected the distribution schedule.
All other issues were connected to missed checks, which may block the contract, and code quality. Nevertheless, all security risk issues were fixed by the team.

The overall security is high enough though the code lacks of readability and the overall quality may be increased. Nevertheless, it performs all desired actions and has solid functionality.

	RATING
Security	9.8
Gas usage and logic optimization	9.5
Code quality	7.5
Test coverage**	10
Total	9.1

** Contracts do not have native unit-test coverage - all tests are written by Blaize Security team in order to achieve sufficient coverage and check business-logic.

COMPLETE ANALYSIS

CRITICAL ✔ Resolved

No decimals specified.

TokenDistributor1.sol, TokenDistributor2.sol
Kale token has a default amount of decimals - 18, however, the amounts of tokens, transferred across contracts TokenDistributor1 and TokenDistributor2, are specified without decimals. As a result, a small dusting amount of tokens would be distributed instead of the full value of tokens.

Recommendation:

Since the Kale token has a default amount of decimals, a key word “ether” can be used to specify the amount of decimals. Example: 4_500_000_000 ether.

CRITICAL ✔ Resolved

Tokens can get stuck on contract in case wallets don’t claim on time.

TokenDistributor1.sol, TokenDistributor2.sol
With the current vesting rules it is assumed that wallets claim their tokens every month. In case, the skipped claiming tokens will get stuck on contract due to restrictions connected with the deadline of vesting for each wallets. As a result, tokens will get stuck on the contract and neither wallet nor the owner will be able to withdraw them.

Recommendation:

Consider creating a mapping to store how much of the tokens have already been released to a specific wallet. After the deadline the rest of required can tokens can be sent to the wallet, so that no tokens will be stuck on smart-contract.

CRITICAL ✔ Resolved

More tokens than allowed are distributed for wallets 2, 3, 4.

TokenDistributor1.sol: function distributeToWallet2(), distributeToWallet3(), distributeToWallet4()
Value from the array “lastClaimTime” is updated before it is checked, whether it is the last claim. Because of this, the portion for the last claim is also distributed at the last but one claim time. For example, as a result, 2_000_000_020 tokens will be distributed for wallet 2 instead of 2_000_000_000.

Recommendation:

Update value from array “lastClaimTime” after if else construction.

HIGH ✔ Resolved

Unverified way of how tokens will appear on distributor contract.

distributor1.sol
All methods (setTokenContract() and all distributeToWallet() methods) rely on the fact, that tokens are already on the contract. Though neither in the constructor nor in the setTokenContract() method there are no checks for token transfer to the contract. Also, since KaleToken has no mint() or mintFor() method and the whole supply is minted to the certain address, there is no guarantee that all tokens will be on the contract for the moment of setTokenContract() call.

Recommendation:

Provide necessary security checks to verify that all tokens are already on the contract at the moment of setTokenContract() call (or during it - for example with transferFrom() the whole supply from the caller).

Post-audit:

Check that tokens are already on distributor1 balance. Also, distributor1 address is provided during deployment of Kale token and supply is minted to the distributor1 balance.

HIGH ✔ Resolved

Not clear schedule

distributor1.sol
The documentation states that the first distribution for all categories is available at day 0, though, all of them (except the category 4) have lastClaimTime (the first time to claim the tokens) after the month (the check for “lastClaimTime[0] <= block.timestamp”). So, there is no ability to claim tokens for these categories at day 0. So, either the documentation, or the contract logic is incorrect.
Also, if following this schedule with the lastClaimTime set initially to 30 days, the last claim in the row will be at the deadline or close to it, thus there will be a problem for the claim.
Also, since the 30 days segments are used for the schedule, it does not conform with the deadline set in years. For example, the first distribution will have 60 distributions by 30 days, which gives 1800 days. Though the deadline is set to 5*365 days, which gives 1825 days. So, there will be 25 days to claim the last distribution for the first category.

Recommendation:

Verify the vesting schedule.
Verify that lastClaimTime should be set to block.timestamp for all categories (or not set at all in setTokenContract()).
Verify, that the deadline is correct and conforms with monthly segments and there will be enough time for the claim of the last distribution.
Verify, that the deadline conforms with the claim times by the number of days.

Post-audit:

First portion of tokens are distributed during setting of contract. In case wallet misses some claims, he is able to claim tokens for missed month.

HIGH ✔ Resolved

First distributed amounts are not subtracted from remaining amounts.

distributor1.sol: function setTokenContract().
Values from array “remainAmounts” are initially set to values from array “DistributorAmount”, however, the first portion of tokens for the first month is distributed instantly in function setTokenContract() and should be subtracted from “remainAmounts” values.

Recommendation:

Subtract amounts, distributed in function setTokenContract() from “remainAmounts” values.

LOW ✔ Resolved

No default visibility for storage varaibles.

distributor1.sol, distributor2.sol
All storage variables have no visibility modifier, thus they all are treated as internal. Since there is no documentation over the variables, and since it is unknown if they should be visible or not, it is recommended to add the visibility.

Recommendation:

Set private/public visibility for the variables.

LOW	✔ Resolved

Addresses not verified.

distributor1.sol, distributor2.sol, constructor.
All addresses of distribution wallets are not verified in any way (neither against zero address, nor against being a contract or any other kind of checks). The issue is set as Low, because all addresses are set in the constructor. Though, since there is no ability to change those addresses, and since there are no tests or deployment scripts to verify those addresses, the issues should be present in the report.

Recommendation:

Provide the verification of the addresses used in the constructor, or add more documentation against these addresses, or add the deployment script to exclude the human error during the deployment.

Post-audit:

Validation, that wallets are not zero addresses, was added. Also, there are now setters for all wallets.

LOWEST	✔ Resolved

Use constants.

distributor1.sol
All numeric values for the distribution amounts (distributeToWallet1() - line 53,; distributeToWallet2() - lines 62, 65; distributeToWallet3() - lines 75, 78; distributeToWallet4() - lines 90, 93; distributeToWallet5() - line 102) should be set as constants in order to increase readability, code quality and ability for further changes.

Recommendation:

Use constants.

LOWEST	✔ Resolved

Extra assignment.

distributor1.sol, distributor2.sol
All addresses are initialized by default values and re-written in the constructor, thus the default initialization (with 0x111 and other values) can be omitted.

Recommendation:

Remove extra initialization.

LOWEST	✔ Resolved

Literal can be used.

KaleToken.sol, constructor()
Literal “ethers” can be used instead of the “* 10**decimals()” construction. decimals() by default is 18 (by OZ ERC20 implementation) thus the number of operations (and gas cost) can be decreased.

Recommendation:

Use “ethers” literal.

LOWEST	✔ Resolved

Requires never revert.

distributor1.sol: functions distributeToWallet1(), distributeToWallet2(), distributeToWallet3(), distributeToWallet4(), distributeToWallet5(). Requires that check Kale token against zero address will never revert. In case token is not set, values from array “deadline” are not set as well and the following functions will fail on requires which check deadline.

Recommendation:

Either remove requires which validate kale token, or put them before requires which validate deadlines, so that there are cases when all requires can revert.

Post-audit:

Validation of kale token are put before other validations.

LOWEST	✔ Resolved

Burnable token.

KaleToken.sol inherits OZ burnable ERC20 standart token. Though, the default burnable functionality allows every user to burn their own tokens. Thus in case of big distributions or any hack for big amount, there is an ability for the malicious user to burn enough tokens to affect the economy of the protocol. Thus verify the usage of the default burn functionality.

Post-audit:

Kale token no longer inherits burnable version of ERC20.

	TokenDistributor1.sol	KaleToken.sol
✓ Re-entrancy	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass
✓ Delegatecall Unexpected Ether	Pass	Pass
✓ Default Public Visibility	Pass	Pass
✓ Hidden Malicious Code	Pass	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass	Pass
✓ External Contract Referencing	Pass	Pass
✓ Short Address/ Parameter Attack	Pass	Pass
✓ Unchecked CALL Return Values	Pass	Pass
✓ Race Conditions / Front Running	Pass	Pass
✓ General Denial Of Service (DOS)	Pass	Pass
✓ Uninitialized Storage Pointers	Pass	Pass
✓ Floating Points and Precision	Pass	Pass
✓ Tx.Origin Authentication	Pass	Pass
✓ Signatures Replay	Pass	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

TokenDistributor2.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	Pass
✔ Default Public Visibility	Pass
✔ Hidden Malicious Code	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass
✔ External Contract Referencing	Pass
✔ Short Address/ Parameter Attack	Pass
✔ Unchecked CALL Return Values	Pass
✔ Race Conditions / Front Running	Pass
✔ General Denial Of Service (DOS)	Pass
✔ Uninitialized Storage Pointers	Pass
✔ Floating Points and Precision	Pass
✔ Tx.Origin Authentication	Pass
✔ Signatures Replay	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Contract: TokenDistributor1

- ✓ Should not set token address if balance of distributor contract was not provided yet (121ms)

Positive use cases:

- ✓ Owner should set new distributor address 1 (148ms)
- ✓ Owner should set new distributor address 2 (109ms)
- ✓ Owner should set new distributor address 3 (112ms)
- ✓ Owner should set new distributor address 4 (105ms)
- ✓ Owner should set new distributor address 5 (95ms)
- ✓ Owner should set new distributor address 6 (105ms)
- ✓ Should set correct token address (88ms)
- ✓ Should distribute to wallet 1 (114ms)
- ✓ Should distribute to wallet 2 (103ms)
- ✓ Should distribute to wallet 3 (103ms)
- ✓ hould distribute to wallet 4 (93ms)
- ✓ Should distribute to wallet 6 (107ms)

Negative use cases: try set old distributor addresses

- ✓ Owner should not set old distributor address 1 to new (90ms)
- ✓ Owner should not set old distributor address 2 to new (82ms)
- ✓ Owner should not set old distributor address 3 to new (83ms)
- ✓ Owner should not set old distributor address 4 to new (88ms)
- ✓ Owner should not set old distributor address 5 to new (78ms)
- ✓ Owner should not set old distributor address 6 to new (79ms)

Negative use cases: try set token address if wallets is zero address

- ✓ Should not set token address if wallet 1 is zero address (54ms)
- ✓ Should not set token address if wallet 2 is zero address (53ms)
- ✓ Should not set token address if wallet 3 is zero address (63ms)
- ✓ Should not set token address if wallet 4 is zero address (50ms)
- ✓ Should not set token address if wallet 5 is zero address (62ms)
- ✓ Should not set token address if wallet 6 is zero address (50ms)

Negative use cases: try distribute to wallets if kale token is zero address

- ✓ Should not distribute to wallet 1 if kale token is zero address
- ✓ Should not distribute to wallet 2 if kale token is zero address
- ✓ Should not distribute to wallet 3 if kale token is zero address
- ✓ Should not distribute to wallet 4 if kale token is zero address
- ✓ Should not distribute to wallet 6 if kale token is zero address

Negative use cases: try distribute to wallets if no time has passed

- ✓ Should not distribute to wallet 1 if no time has passed (85ms)
- ✓ Should not distribute to wallet 2 if no time has passed (77ms)
- ✓ Should not distribute to wallet 3 if no time has passed (83ms)
- ✓ Should not distribute to wallet 4 if no time has passed (81ms)
- ✓ Should not distribute to wallet 6 if no time has passed (77ms)

Positive use cases: distribute to wallets if too much time has passed

- ✓ Should not distribute to wallet 1 if too much time has passed (103ms)
- ✓ Should not distribute to wallet 2 if too much time has passed (85ms)
- ✓ Should not distribute to wallet 3 if too much time has passed (101ms)
- ✓ Should not distribute to wallet 4 if too much time has passed (87ms)
- ✓ Should not distribute to wallet 6 if too much time has passed (84ms)

Negative use cases: try distribute to wallets if wallets is zero address

- ✓ Should not distribute to wallet 1 if wallet 1 is zero address (100ms)
- ✓ Should not distribute to wallet 2 if wallet 2 is zero address (89ms)
- ✓ Should not distribute to wallet 3 if wallet 3 is zero address (97ms)
- ✓ Should not distribute to wallet 4 if wallet 4 is zero address (90ms)
- ✓ Should not distribute to wallet 6 if wallet 6 is zero address (93ms)

Contract: TokenDistributor2

- ✓ Should set correct token address
- ✓ Owner should set new distributor address 1
- ✓ Owner should not set old distributor address 1 to new
- ✓ Owner should set new distributor address 2
- ✓ Owner should not set old distributor address 2 to new

- ✓ Owner should set new distributor address 3
- ✓ Owner should not set old distributor address 3 to new
- ✓ Owner should set new distribution option
- ✓ Should correct distribution to wallets if distribution option = 1 (standard setting) (66ms)
- ✓ Should correct distribution to wallets if distribution option = 2 (custom setting) (63ms)
- ✓ Should revert distribution to wallet if kale token not set
- ✓ Should revert distribution to wallet if wallet1 not set
- ✓ Should revert distribution to wallet if wallet2 not set
- ✓ Should revert distribution to wallet if wallet3 not set
- ✓ Not owner should not call onlyOwner functions (60ms)

60 passing (5s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
KaleToken.sol	100	100	100
TokenDistributor1.sol	100	88.37	100
TokenDistributor2.sol	100	100	100
All files	**100**	**90.2**	**100**

DISCLAIMER

The information presented in this report is an intellectual property of the customer including all presented documentation, code databases, labels, titles, ways of usage as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else requirements and be fully secure, complete, accurate and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.