PUBLIC

Code Assessment

of the Dual Token Smart Contracts

December 15, 2025

Produced for

BLOCK V / DUAL

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help BLOCKv with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Dual Token according to Scope to support you in forming an opinion on their security risks.

BLOCKv implements a migration system to convert the BLOCKv token (VEE) to the new Dual token (DUAL).

The most critical subjects covered in our audit are access control, functional correctness, and asset solvency. Security regarding all the aforementioned subjects is high.

The general subjects covered are documentation, testing, gas efficiency, and Solidity compiler version specification. Documentation and testing are satisfactory. All identified issues in the remaining areas were addressed after the intermediate report.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Dual Token repository based on the documentation file `README.md`. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	18 Nov 2025	cfc077364d6265c804aacbc3b6089a7658128bb2	Initial Version
2	12 Dec 2025	f9c8ff70c93428fd011c4b788cb477afd3f6f703	Version After Fixes

The Solidity smart contracts use a floating pragma (^0.8.27), allowing compilation with any Solidity 0.8.x version from 0.8.27 onward. After the intermediate report the compiler version has been fixed to `0.8.27`.

The following files are in scope:

- `code/dual-token/src/Dual.sol`
- `code/dual-token/src/Migrator.sol`
- `lib/openzeppelin-contracts/contracts/finance/VestingWallet.sol` (OpenZeppelin version 5.4.0 at `c64a1edb`)

2.1.1 Excluded from scope

Any contract not explicitly in scope is to be considered out of scope.

The 1:2 migration exchange rate combined with DUAL's larger total supply results in dilution. The economic effects are out of scope.

The deployment of the new Dual token, the migrator contract, and the vesting wallet is assumed to be performed correctly.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

BLOCKv offers a migration system to convert the BLOCKv token (VEE) to the new Dual token (DUAL).

The system consists of three main components:

1. **VEE Token**: The legacy token being migrated.
2. **Migration Agent Contract**: The `Migrator.sol` contract, responsible for managing the migration process.
3. **DUAL Token**: The new token replacing VEE. Existing holders can migrate their VEE to DUAL. Defined in `Dual.sol`.

2.2.1 VEE Token

The VEE Token is the legacy ERC20 token that users migrate from. It is deployed at the address `0x340D2bdE5Eb28c1eed91B2f790723E3B160613B7` and includes the following migration functions:

- `setMigrationAgent()`: This function can only be called by the migration master, and it sets the address of the migration agent contract.
- `migrate()`: This function can be invoked by any user to burn their VEE tokens and trigger the `migrateFrom()` function in the migration agent contract, which enables the conversion to DUAL tokens.

2.2.2 Migration Agent (Migrator)

The **Migration Agent** is defined in the `Migrator.sol` contract, and it handles the actual migration process.

The migration begins at deployment and has a maximum duration of two years. There is a cap on the total DUAL tokens available for migration, set to twice the VEE total supply. This cap must be funded at deployment and—given the 1:2 exchange rate—allows all VEE tokens to be migrated.

When the migration period ends, or if the migration cap is reached, the owner can withdraw any remaining DUAL to the beneficiary. The beneficiary is set to the owner at deployment but can be updated.

The key functions of the migrator are:

- `migrateFrom()`: This function can only be called by the VEE token contract. It verifies if the current time falls within the migration period, computes the amount of DUAL tokens to issue (2 DUAL tokens for each 1 VEE token burned), and transfers the corresponding DUAL tokens to the recipient.
- `finalizeMigration()`: This function can only be called by the owner. After checking the finalization requirements, it sets the migration status to finalized, preventing further calls to `migrateFrom()`. In addition, it transfers any leftover Dual balance to the `beneficiary`.
- `setBeneficiary()`: This function can only be called by the owner. It sets the beneficiary of the leftover Dual.

2.2.3 Dual Token

The **DUAL Token** represents the new token that users migrate to. This ERC20 token also includes permit functionality, and the total supply is 10 billion tokens. Initially, the entire supply is minted to the *initial Dual holder* (the `mintTo` address), as specified in the constructor.

The distribution plan for DUAL tokens after deployment is as follows:

- **Migration Reserve**: Approximately 7.2 billion DUAL tokens will be reserved for the migration, that is equal to the VEE total supply multiplied by 2.
- **Liquidity Pool Vesting**: 1 billion DUAL tokens will be vested linearly over one year to support liquidity.
- **Admin Allocation**: Any remaining tokens after the migration and vesting allocations will go to the admin.

2.2.4 Vesting Wallet

The **Vesting Wallet** contract is responsible for releasing DUAL tokens to the admin over time. It works as follows:

- **Linear Vesting Schedule**: DUAL tokens are locked in the vesting wallet and can be released gradually based on a linear schedule over one year.

- **Release Mechanism**: Anyone can call the `release()` function to transfer tokens to the wallet owner (the admin) according to the vesting schedule.

The wallet supports vesting both native currency and ERC20 tokens, however, this project only uses the vesting wallet with Dual tokens.

2.3 Changes in V2

The distribution plan for DUAL tokens was changed as follows:

- **Migration Reserve**: Unchanged; Approximately 7.2 billion DUAL.
- **Liquidity Vesting Pool 1**: the vesting period was changed from 1 year to 5 years. The same amount (1 billion DUAL) is allocated.
- **Liquidity Vesting Pool 2**: this substitutes the previous Admin Allocation, and consists of all the remaining token after the migration and the vesting in the first pool. This amount (around 1.7 billion DUAL) will be vested linearly over 5 years.

2.4 Trust Model

- Users: untrusted; can perform arbitrary actions.
- Migration Master: fully trusted; can set the migration agent address. The migration reserve will be transferred to this address. A malicious or compromised migration agent could cause a significant loss of funds.
- Initial DUAL holder (`mintTo` address): fully trusted; receives the total supply of DUAL on deployment.
- Admin: fully trusted; owns the migration agent and receives the liquidity pool vesting and the admin allocation.

2.4.1 Changes in V2

- The Admin Allocation was removed. As a consequence, the admin does not receive the Admin Allocation, which is instead vested in the Liquidity Vesting Pool 2 described above.
- VESTING_POOL_1_BENEFICIARY and VESTING_POOL_2_BENEFICIARY: fully trusted; receive the allocations in the Liquidity Vesting

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0
Informational Findings	2

- Compiler Version Is Not Fixed Code Corrected
- Gas Optimizations Code Corrected

6.1 Compiler Version Is Not Fixed

Informational Version 1 Code Corrected

CS-BLKV-D-003

The Solidity source files use a floating pragma statement (`^0.8.27`) and the `foundry.toml` file does not pin a specific compiler version either. To ensure consistent behavior and avoid unexpected issues, contracts should be compiled and deployed using a fixed compiler version and dependency set that have been thoroughly tested together.

Code corrected:

BLOCKv has fixed the compiler version by specifying it inside the `foundry.toml` file.

6.2 Gas Optimizations

Informational Version 1 Code Corrected

CS-BLKV-D-002

Several parts of the implementation could be optimized to reduce gas consumption, including but not limited to:

1. In `migrateFrom()`, the zero-check for `value` is redundant since the function can only be called from `VEE.migrate()` which already checks `value`.
2. The function `setBeneficiary()` performs a redundant storage read of the variable `beneficiary`.
3. The function `finalizeMigration()` performs a redundant storage read of the variable `beneficiary`.

Code corrected:

BLOCKv has updated the code to optimize all of the points mentioned above.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 VestingWallet Comment References EOA-to-contract Migration as Future Possibility

Informational | Version 1 | Acknowledged

CS-BLKV-D-001

The VestingWallet contract comment states: "there is likely to be a migration path for EOAs to become contracts in the near future."

EIP-7702 went live on Ethereum mainnet on May 7, 2025 as part of the Pectra upgrade. EOAs can now delegate to smart contracts, making them function like contract accounts. This migration path is no longer a future possibility but an active reality.

The comment should be updated to reflect current Ethereum capabilities.

Acknowledged:

BLOCKv is aware of the outdated comment and has decided to leave it unchanged.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 BLOCKv Allows Setting `migrationAgent` Only Once

Note Version 1

The BLOCKv token allows the migration master to set the `migrationAgent`. This can only be done once, as it requires no migration agent to be set prior. Hence this action is irreversible and must be executed carefully.

8.2 BLOCKv Must Not Be Paused Prior to Migration Completion

Note Version 1

The BLOCKv token features a pausing functionality for transfers. This must not be done while migration is active, to not interrupt it. While the migration function itself is unaffected by the pause functionality, non-EOA accounts may need to transfer tokens to an account prior to being able to call `migrate()`. A paused token contract would inhibit them from migrating.

8.3 Migration Must Be Initiated by Token Holders

Note Version 1

All token holders of BLOCKv must initiate the migration of their tokens manually prior to the migration period ending (2 years) by calling `migrate()` on the token contract.

Smart contracts holding these tokens are unable to do this on their own. Tokens are expected to be transferred to an address able to execute this.

Several smart contracts hold this token, including Uniswap pools and escrow contracts for bridges. All these tokens must be manually converted within the migration timeframe; for bridges this means bridging back to L1 and performing the conversion, whereas the Uniswap pool needs to be wound down to migrate the tokens.

8.4 Renouncing the Vesting Wallet Ownership Permanently Locks Funds

Note Version 1

The beneficiary of the vested funds in the `VestingWallet` is the owner of the contract. The `VestingWallet` contract inherits from `Ownable`, which exposes the `renounceOwnership` function by default. If the owner renounces ownership, all remaining funds (both vested and unvested) will be

permanently locked in the wallet, as the release functions attempt to transfer funds to the owner address, which would be `address(0)` after renunciation.