Blaize.Security

ĀUT LABS
SMART CONTRACT AUDIT

December 4th, 2024 / V. 1.0

Table of Contents

Executive Summary

During the audit, we examined the security of the smart contracts for the Āut Labs protocol. Our task was to identify and describe any security issues within the platform's smart contracts. This report presents the findings of the security audit of the **Āut Labs** smart contracts conducted between **November 6th** and **November 26th**.

Blaize.Security conducted an in-depth audit of the Āut Labs smart contracts. These contracts form the core of a decentralized platform utilizing ERC20 tokens—Aut and c-aut — for various purposes such as token vesting and interactions within a plugin ecosystem. The audit focused on the security and functionality of key features, including token distribution mechanisms, vesting schedules, role-based access controls, and the integration of upgradeable contracts. We also examined centralization aspects and the robustness of the deployment script. The Protocol Overview section provides further details on the contract logic and our recommendations.

The security team decomposed the protocol, verified the integrity of the business logic, funds flow, access control systems, and user workflows. Also was analyzed a set of edge cases and performed extensive testing. All issues identified during the process are described in the Complete Analysis section. The Āut Labs team resolved or acknowledged a significant portion of these issues. However, the report still contains unresolved concerns related to centralization risks and adherence to best practices in code structure and deployment procedures.

The security team list several notes to available in the report:

1. ERC20 implementation
   Aut implements standard ERC20 interface (OZ implementation), thus the security team verifies the correctness of the implementation.

2. Aut distribution.

The Distributor contract expects that the whole AUT supply will be transferred directly to the contract. However the whole supply is initially minted to the deployer of the token.

Thus the system will have an intermediate step between the token creation and its distribution. While it is an adopted practice and the deployment script handles this step, it is still a point for additional monitoring. As in case of incorrect transfer to the contract, the whole transferred supply will be effectively locked - as the contract expects full supply to be transferred. The same applies to vesting contracts - they expect Aut token to be transferred directly to the contract, thus it is a point for monitoring.

3. Token distribution should be curated

Currently the Distributor contract introduces no validations for addresses, thus there are no checks if they are different (except the contracts used). And in general the validation of addresses and all sanitizing actions are curated by the team. Currently addresses are curated in the deployment script - however the deployment script currently get necessary addresses from env variables which cannot be evaluated at the moment. However the team plans to have separate Gnosis Multisig for each distribution part as a temporary storage. Also it should be noted, that the cliff period is set manually for each vested group. The deployment script has combined procedure with deployment of contracts which are still under development - though no cross-influence detected, still security team advises to separate deployment flows. The security team advises to have these items as a point of monitoring and re-check it before the production stage.

4. Partial release

The logic of the contract allows a user to have a partial release of tokens from vesting (claim only part of tokens available for claiming). Thus it requires the user to pass a correct amount into the interface. Usually the vesting contract releases the whole amount available at the moment (whole claimable amount), as otherwise it may create complications for the UI and unnecessary requirements for the user (e.g. self-calculation of tokens). While there is no issue connected to the implementation and the partial release is valid, it is a note worth to be mentioned in the report.

The security team currently evaluates the project as **Secure**. The Aut team resolved all issues, and the security team notes good approach in token deployment and funds distribution. Though the security team notes this points as a subject for additional monitoring (refer to the connect parts in the protocol description section). Also, the team verified contracts connected to the main protocol token distribution, with additional functionality (cAut and Reputation mining) left for the next iteration.

Auditing strategy and Techniques applied/Procedure

Blaize.Security auditors start the audit by developing an auditing strategy - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

MANUAL AUDIT STAGE

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

and a wide spectrum of other vulnerable areas.

FOR ADVANCED COMPONENTS:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

TESTING STAGE:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

POST-AUDIT STEPS RECOMMENDED

To ensure the security of the contract, the **Blaize.Security** team suggests that the team follow post-audit steps:

1. Request audits of other protocol components (dApp, backend, wallet, blockchain, etc) from Blaize Security
2. Request consulting and deployment overwatch services provided by Blaize Security
3. Launch active protection over the deployed contracts to have a system of early detection and alerts for malicious activity. We recommend the AI-powered threat prevention platform **VigiLens**, by the **CyVers** team.
4. Launch a **bug bounty program** to encourage further active analysis of the smart contracts.
5. Request post-deployment assessment service provided by Blaize Security to ensure the correctness of the configuration, settings, cross-connections of deployed entities and live functioning

Audit Rating

Score: **10** /10

	RATING
Security	10
Logic optimization	9.9
Code quality	10
Testing suite	8.5
Documentation	9.9

Security: General mark for the security of the protocol.
The main mark for the audit qualification.

Logic optimization: Evaluation of how optimal the implementation is, including presence of extra/unused code, uncovered/extra cases, gas (or its analog) optimization, memory management optimization, etc
Code quality: Evaluation of best practices followed, code readability, structure and convenience of further development
Testing suite: Availability of the native tests suite, level of logic coverage, checks of critical areas being covered.
Documentation: Availability and quality of the documentation, coverage of core functionality and user flows: whitepaper, gitbook, readme, specs, natspec, comments in the code and other possible forms of documentation.

SECURITY RATING CALCULATION

Approximate weight of unresolved issues.
Critical: -3 points
High: -2 points
Medium: -0.5 points
Low: -0.1 points
Informational: -0.1 point (in general, depends on the context)

Note: additional concerns, violated checklist items (including standard vulnerabilities), and verified backdoors may influence the final mark and weight of certain issues.

Starting with a perfect score of 10:

Critical issues: 0 issue (0 resolved): 0 points deducted

High issues: 0 issue (0 resolved): 0 points deducted

Medium issues: 0 issues (0 resolved): 0 points deducted

Low issues: 2 issues (2 resolved): 0 points deducted

Informational issues: 3 issues (3 resolved): 0 points deducted

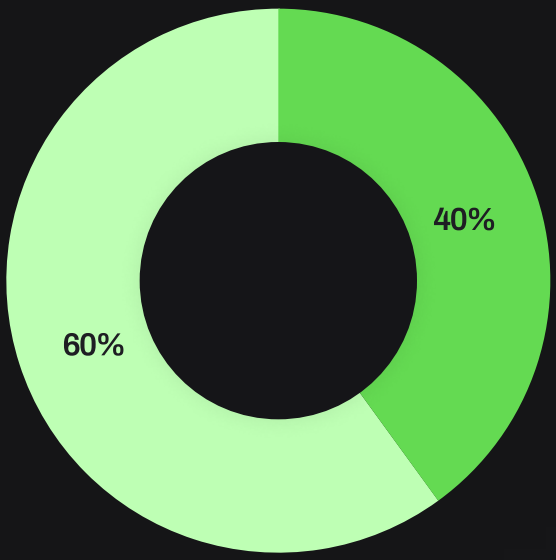
Other: -

Security rating = 10

Technical Summary

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

Critical
High
Medium
Low
Info



The table below shows the number of the detected issues and their severity. A total of 5 problems were found. 5 issues were fixed or verified by the Customer's team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	2	2
Info	3	3

Best practices and optimizations

- 12 items resolved
- 2 items unaddressed but verified

Section is marked as **resolved**

SEVERITY DEFINITION

CRITICAL

The system contains several issues ranked as very seriousand dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

HIGH

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

MEDIUM

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

LOW

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

INFO

The issue has no impact on the contract's ability to operate, yet is relevant for best practices. Or this status can be assigned to the issues related to the suspicious activity or substandard business logic decisions which cannot be classified without the comments from the team (and can be re-classified on the later audit stages).

Issues reviewed by the team can get the next statuses:
Resolved: issue is resolved by an appropriate patch or changes in the business logic
Verified: the team provided sufficient evidences that the issue describes desired behavior
Unresolved: neither path nor comments provided by the team, or they are not sufficient to resolve the issue
Acknowledged: the team accepts the misbehavior and connected risks

Audit Scope

Language/Technology: **Solidity**

Blockchain: **Ethereum**

The scope of the project includes:

- contracts/repfi/token/Distributor.sol
- contracts/repfi/token/REPFI.sol
- contracts/repfi/vesting/TokenVesting.sol
- contracts/repfi/vesting/EarlyContributors.sol
- contracts/repfi/vesting/FounderInvestors.sol
- contracts/repfi/vesting/KOLsAdvisors.sol

Note: initial scope also included overview of cAut and Reputation Mining contracts, though these contracts are moved to the next audit iteration and are excluded from this report.

Repository: https://github.com/Aut-Labs/contracts/tree/repfi

The source code of the smart contract was taken from the branch:
repfi.

Initial commit:

- 75831829c1d804005faea73be5a5d855ed4eddb9

Final commit:

- b1c67fa29ef95e85804a9d568fe495e6c67a9b7a

Protocol overview

DESCRIPTION

The CAut contract represents the Conditional Aut token (symbol c-aut), a custom ERC20 token used within the platform. This token integrates controlled transfer capabilities to ensure it can only be transacted or approved through registered plugins defined in the UtilsRegistry contract.
Note: During the audit, the contract name was changed from CrepFi to CAut and was moved to the next audit iteration.

The Distributor contract is responsible for the initial distribution of Aut tokens across different stakeholders and contracts. It allocates specific amounts of tokens to designated addresses and contracts, such as sale, reputation mining, airdrop, founder investors, early contributors, listing, kolsAdvisors and the treasury.
Note: during audit: founder investors <- investors, early contributors <- teams, listing <- partners, kolsAdvisors was added during audit, treasury <- ecosystem fund.

The Aut contract represents the Aut token, an ERC20 token with a fixed supply created to support the platform's ecosystem.
Note: During the audit, the contract name was changed from RepFi to Aut.

The UtilsRegistry contract acts as a registry for plugins within the Aut ecosystem, managing the plugins authorized to interact with the c-aut token. This contract restricts c-aut transactions to registered plugins, enhancing security and control over token movement.
Note: The contract was moved to the next audit iteration.

The TokenVesting contract is designed to manage token vesting schedules, allowing for controlled token release over time based on a pre-set configuration. Flexibility is provided through customizable settings established during initialization, such as vesting duration, release intervals, and revocability.

The KOLsAdvisors contract is a token vesting contract structured to handle the vesting of tokens for investors. It extends the TokenVesting contract, setting specific parameters for the vesting schedule, amount, and revocability.
Same description for FounderInvestors and EarlyContributors contracts.

ROLES AND RESPONSIBILITIES

1. **CAut** (Conditional Aut Token):
The contract implements the standard OZ Access Control system
a. Admin (DEFAULT_ADMIN_ROLE): Can add or remove roles, including the BURNER_ROLE. Manages overall permissions for the contract.
Note: by default, the admin is assigned to the designated address from the constructor parameter
b. Burner (BURNER_ROLE): Has the right to burn tokens on behalf of users, helping control the overall token supply.
c. Plugin: Registered plugins act as senders (transfer()) and receivers of tokens (transferFrom), controlled through the UtilsRegistry contract.
d. Regular User: Can receive tokens from a plugin and send tokens to a plugin.
e. Deployer: receives the initial supply of the token
Note: During the audit, the contract name was changed from CrepFi to CAut and was moved to the next audit iteration.

2. **Distributor**:
a. Owner: The only entity that can initiate the distribution (distribute) of tokens to various recipients. By default it is set to the contract's deployer.
b. Sales Multisig (sale), Reputation Mining Contract, Airdrop Contract (airdrop), Founder Investors, Early Contributors, Listing, Treasury and Kols Advisors: receive their designated tokens after distribution.

3. **Aut** (Āut Labs Token):
a. Deployer (Owner): Initially receives the full token issuance (100 million tokens), which can then be allocated as per contract specifications or for various purposes.
Note: During the audit, the contract name was changed from RepFi to Aut.

5. UtilsRegistry (Utils Registry):
a. Owner: The only entity that can add (registerPlugin) or remove (removePlugin) plugins, controlling which contracts are allowed to interact with the c-aut token.
b. Plugin: Registered plugins are allowed to receive and transfer c-aut tokens.

6. TokenVesting, Team Vesting Contract, Investors Vesting Contract
a. Owner (based on OZ Ownable approach): Creates vesting schedules, adds new recipients, and sets parameters for each schedule. Can revoke schedules if they are revocable. Has the right to withdraw unallocated funds from the contract.
Note: Can release vesting funds on behalf of the beneficiary.
b. Beneficiary: Receives tokens according to the vesting schedule. Can release vested tokens upon meeting vesting conditions.
Note: Beneficiary can have several vestings.

LIST OF VALUABLE ASSETS

1. **Aut Token (ERC20):**

Purpose: The main ecosystem token, used for rewards, vesting, and distribution among users, investors, the team, and for ecosystem and partner support.
Usage in Contracts:
Distributor (distributes tokens to various addresses and contracts)
Note: the contract expects that the Aut supply will be transferred directly to the contract.
ReputationMining (allocates tokens to users based on their activity)
Investors Vesting Contract (designated for investor vesting) - receives tokens during the distribution.

Team Vesting Contract (designated for team vesting) - receives tokens during the distribution.
TokenVesting (used for flexible vesting arrangements)
Note: System expects that the Aut supply will be transferred directly to the contract.
Note: The whole supply (100 mln) tokens are minted during the deployment for the deployer.

General warning: Neither of contracts implements rescue mechanism and no other tokens are expected to be supported. Thus any token other than Aut will be locked on the contract. And since contracts expect direct transfers of tokens - transferred amounts should be curated.

DEPLOYMENT

1. **Transaction Success Checks:**

The deployment script doesn't check the success of certain important transactions, such as autId.setHubRegistry() and autId.transferOwnership(). It's recommended to add error handling for these calls so that the script terminates with an error message if something fails. This will help avoid cases where the contract deploys but doesn't initialize correctly.

2. **Enhanced Logging:**

While the script already uses console.log, additional logs can be added at key steps, such as contract deployments and initializations. This will give a clearer view of the deployment process, which is especially helpful when deploying multiple contracts. For instance, logging initial parameters during deployment and indicating the successful completion of each step can provide valuable feedback.

3. Recommendation to Use OpenZeppelin Upgrades for Foundry:

It's recommended to integrate OpenZeppelin Upgrades for Foundry, which supports working with proxy contracts within the Foundry framework. This can simplify managing and upgrading contracts and ensure a more secure deployment and testing process. Using this plugin avoids manually writing proxy logic and improves compatibility with OpenZeppelin's current security standards for upgradable contracts.
https://docs.openzeppelin.com/upgrades-plugins/1.x/foundry-upgrades

4. Deployment script has all necessary parameters to distinguish mainnet and testnet deployments. All parameters are set as env variables, thus it is a point of additional monitoring during the deployment. However the script has combined procedure with deployment of contracts which are still under development - though no cross-influence detected, still security team advises to separate deployment flows.

CONFIGURATION AND SETTINGS

Aut (Aut Labs Token)

Initialization Settings:

1. Token Name: “Aut Labs”.
2. Token Symbol: “AUT”.
3. Total Supply: Set to 100 million tokens, minted at deployment to the deployer.

Distributor

Initialization Settings:

1. aut (IERC20): Address of the Aut token contract.
2. sale (address): Address for the sales wallet.
3. reputationMining (IReputationMining): Address of the ReputationMining contract.
4. airdrop (address): Address of the Airdrop contract.
5. founderInvestors (TokenVesting): Address of the founder investors’ TokenVesting contract.
6. earlyContributors (TokenVesting): Address of the early contributors TokenVesting contract.
7. treasury (address): Address for the treasury wallet.
8. kolsAdvisors (TokenVesting): Address of the kols advisors TokenVesting contract.

Note: owner of the contract is set to the contract deployer.

EarlyContributors and FounderInvestors Vesting Contracts

Initialization Settings:

1. token_ (IERC20): Address of the ERC20 token used for vesting (usually Aut).
2. _owner (address): Address of the contract’s owner with administrative control.

Fixed Parameters:

1. DURATION (uint256): Vesting duration (12 months).
2. RELEASE_INTERVAL (uint256): Interval for token releases (30 days).
3. REVOCABLE (bool): Indicates whether the vesting is revocable (set to true).

KOLsAdvisors Vesting Contract

Initialization Settings:

1. token_ (IERC20): Address of the ERC20 token used for vesting (usually Aut).
2. _owner (address): Address of the contract's owner with administrative control.

Fixed Parameters:

1. DURATION (uint256): Vesting duration (6 months).
2. RELEASE_INTERVAL (uint256): Interval for token releases (30 days).
3. REVOCABLE (bool): Indicates whether the vesting is revocable (set to true).

TokenVesting

Initialization Settings:

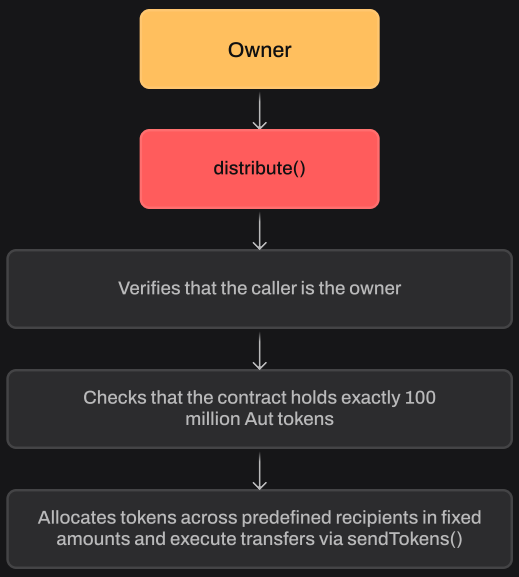
1. token_ (IERC20): Address of the ERC20 token used for vesting.
2. owner (address): Address of the contract's owner with authority over vesting schedules.
3. duration (uint256): Duration of the vesting period.
4. releaseInterval (uint256): Interval between token releases.
5. revocable (bool): Indicates whether vesting can be revoked.

Fixed Parameters:

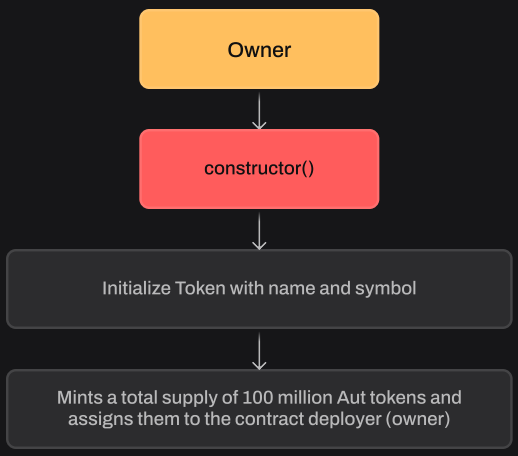
1. MAX_ARRAY_LENGTH (uint256): Maximum number of recipients that can be added per call (100).

ĀUT LABS

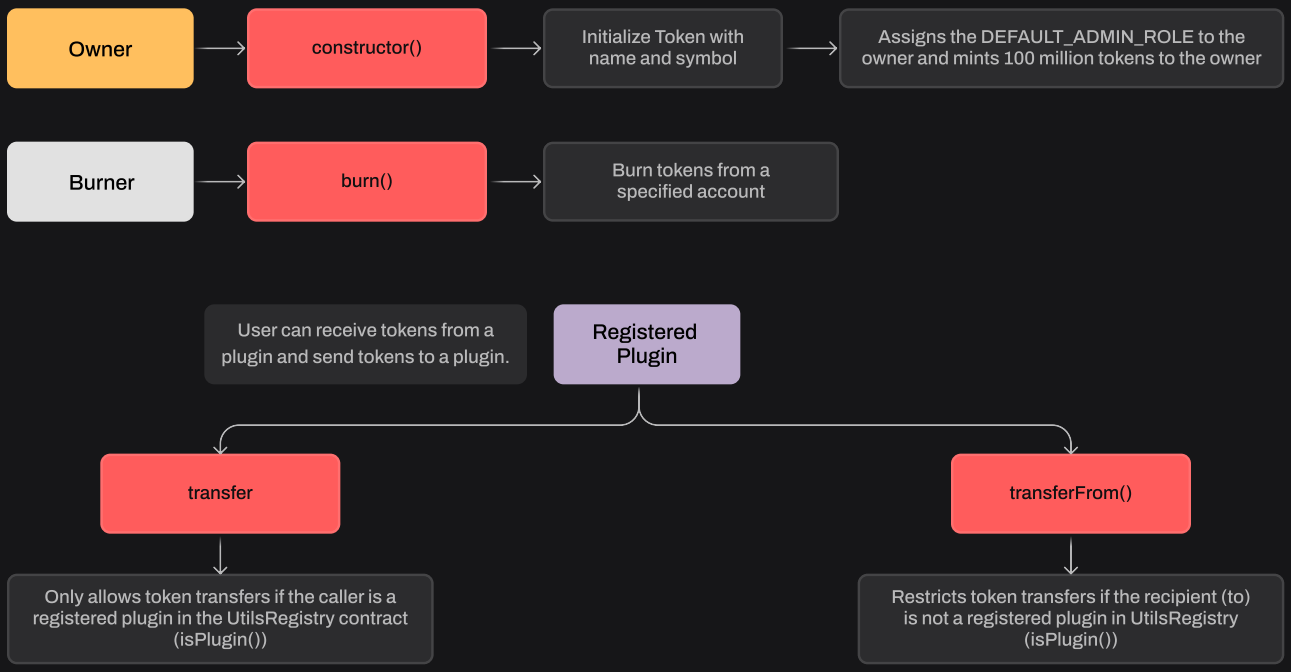
Distributor



AUT



CAUT (ERC20 version - will be changed in the next iteration)

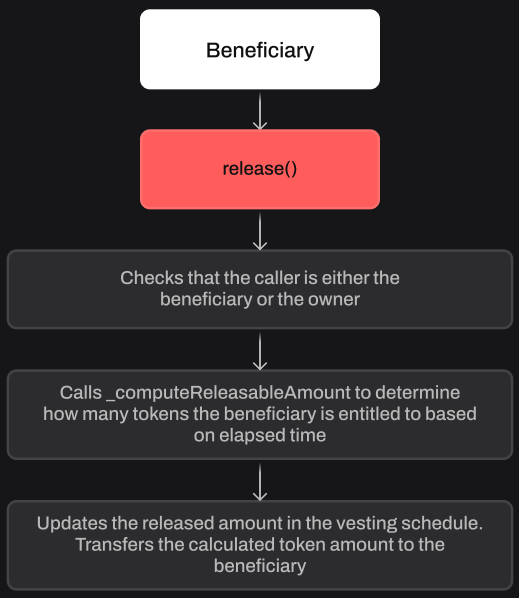


ĀUT LABS

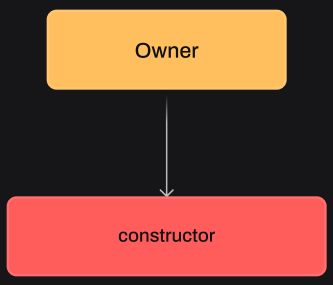
TokenVesting

ĀUT LABS

TokenVesting



Investors, Team

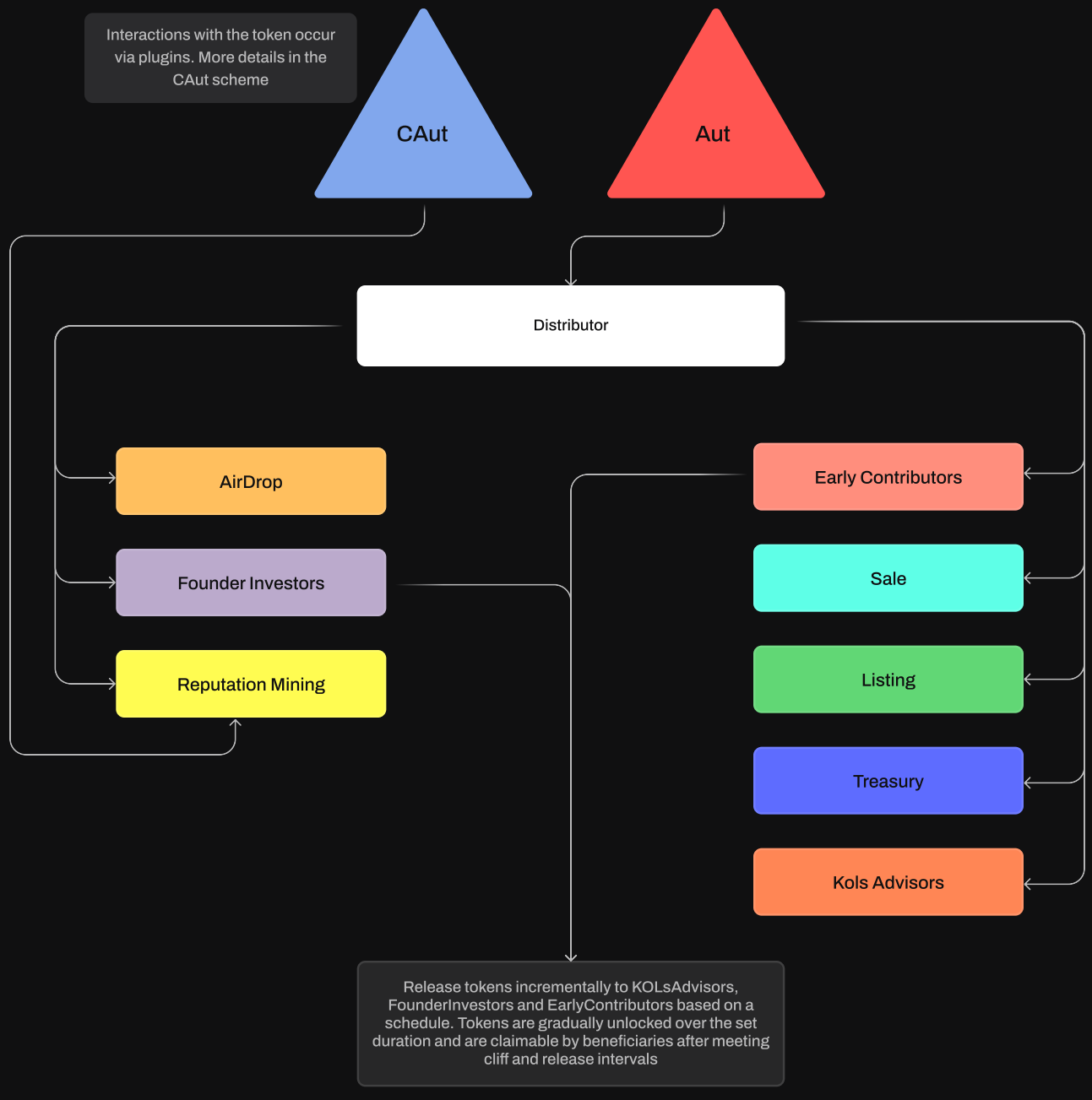


1. **Token Address** (token_): Address of the ERC20 token used for vesting.
2. **Owner Address** (_owner): Owner of the contract.
3. **Duration** (DURATION): Vesting duration, set to 6 months.
4. **Release Interval** (RELEASE_INTERVAL): Interval between token releases, set to 30 days.
5. **Revocable** (REVOCABLE): Indicates that vesting is revocable.

The **Team** and **Investors** contract inherit the main functionality of **TokenVesting** and apply it specifically to team/investors members with pre-set vesting parameters.

ĀUT LABS

Token Movement



Complete Analysis

STANDARD CHECKLIST / VULNERABLE AREAS

Storage structure and data modification flow	Pass
Access control structure, roles existing in the system	Pass
Public interface and restrictions based on the roles system	Pass
General Denial Of Service (DOS)	Pass
Entropy Illusion (Lack of Randomness)	N/A
Order-dependency and time-dependency of operations	Pass
Accuracy loss, incorrect math/formulas other violated operations with numbers	Pass
Validation of function parameters, inputs validation	Pass
Asset management, funds flow and assets conversions	Pass
Signatures replay and multisig schemes security	N/A
Asset Security (backdoors connected to underlying assets)	Pass
Incorrect minting, initial supply or other conditions for assets issuance	Pass
Global settings mis-using, incorrect default values	Pass
Violated communication between components/modules, broken co-dependencies	N/A
3rd party dependencies, used libraries and packages structure	Pass
Single point of failure	Pass
Centralization risk	Pass
General code structure checks and correspondence to best practices	Pass
Language-specific checks	Pass

LOW-1 Resolved

UNNECESSARY MODIFIER ON THE VIEW METHOD.

TokenVesting.sol: computeReleasableAmount()
The function has a modifier which reverts to the revoked vesting. However, the function already implements logic to return 0 for the revoked vestings. Thus the revert will create a confusing situation for the viewer connected to the dApp and will cause unnecessary errors for the dApp.
Additionally, the revert introduces no error, overcomplicating the debug.
The issue is marked as Low, as while there no vulnerability introduced but the issue influences the dApp healthy behavior.

RECOMMENDATION:

Remove unnecessary modifier and add the error to the revert in it.

LOW-2 Resolved

ALLOCATION SUM VALIDATION.

TokenVesting.sol: addRecipients().
In the addRecipients function of the TokenVesting contract, there is no check to ensure that the sum of all allocations equals 100% (i.e., the value of the PERCENTAGE_DENOMINATOR). This oversight can lead to errors if the total allocation does not match the intended amount. If an incorrect allocation is provided, the function will revert when checking balances, causing confusion for the caller due to an unclear error source.

RECOMMENDATION:

Add a validation step to check that the sum of the _allocations array equals the PERCENTAGE_DENOMINATOR before processing any recipient allocations. This will provide a clear error message if allocations are incorrectly configured, improving user experience and reducing potential confusion.

INFO-1 Resolved

PRESENCE OF TODOS/COMMENTS INDICATING POTENTIAL CODE CHANGES.

1. Team.sol, Investors.sol: TOTAL_VEST_AMOUNT;

The variable TOTAL_VEST_AMOUNT is defined but appears to be unused in the contract. Additionally, the comment "// may not be needed as we're using the balance to check this in TokenVesting" is ambiguous and raises uncertainty about the necessity of the variable.

Confirm whether the TOTAL_VEST_AMOUNT variable is essential for the contract's logic. If the variable is redundant, it should be removed to reduce unnecessary complexity. If the variable is necessary, provide clear documentation on its intended purpose.

2. TokenVesting.sol

withdraw() function, comments on the target for transfer

The issue is marked as info, as it requires additional information from the team. However, such comments may signalize on the unfinished or obsolete functionality, thus the criticality may be changed based on the feedback.

RECOMMENDATION:

Resolve todo's and analog comments, finalize the functionality or verify the current state of marked places.

POST-AUDIT:

1. Unused Variable - TOTAL_VEST_AMOUNT in Team.sol and Investors.sol: The variable was deleted.
2. Withdraw Function in TokenVesting.sol: Comment regarding the target address for transfers was removed.

INFO-2 Resolved

PAYABLE KEYWORD USAGE IN ERC-20 TRANSFER.

TokenVesting.sol: release().
In the release function, the beneficiary address is cast to address payable, although the function only transfers ERC-20 tokens, not native ETH. The payable keyword is generally intended for addresses that will receive native currency transfers, so its necessity here is unclear.

The issue is marked as info as requires additional information from the team.

RECOMMENDATION:

Verify whether the payable keyword is needed here. If this is an oversight, remove payable casting from beneficiaryPayable and use a simple address instead.

POST-AUDIT:

Payable casting was removed.

INFO-3 Resolved

DISTRIBUTION DOES NOT MATCH THE DOCUMENTATION

Distributor.sol. The contract states the next distribution:

- Sale: 15M
- Reputation Mining: 36M
- Airdrop: 4M
- Investors: 10M
- Team: 5M
- Partners: 10M
- Ecosystem: 20M

However, according to the documentation (located in the notion) the distribution should look like:

- Sale: 7M
- Reputation Mining: 36M
- Airdrop: 4M
- Investors (Founder/Shareholders): 10M
- Team: 5M
- Treasury (Ecosystem and Partners): 25M
- Advisors: 5M
- Listing/Marketing: 8M

Therefore some numbers (for Sale) and some categories differ. The security team understands, that the tokenomics of the project may evolve with time. Therefore the issue is marked as Info - as more information is required to assess which version of distribution (in code or in doc) is outdated and requires the update of numbers.

RECOMMENDATION:

Verify the correct version of the tokenomics and correct the numbers either in the code or in the documentation

Resolved

BEST PRACTICES AND CODE STYLE VIOLATIONS, OPTIMIZATIONS

1. Unnecessary imports

Distributor.sol: TokenVesting import and IReputationMining import

The contract imports TokenVesting and declares 2 variables with it. However, the contract is not used. Thus it should be removed to decrease the size of the deployed bytecode and save gas spendings. The same applies to the imported interface.

The same applies to the SafeERC20 import - the contract utilizes it for the safeTransfer() function. However, the Aut token implements standard OZ ERC20 interface, thus the SafeERC20 library is not necessary for this case. While no security issue was introduced, it just increased gas spendings during the distribution.

2. Zero-address validation.

Distributor.sol: constructor();

TokenVesting.sol: _createVestingSchedule() -> _beneficiary;

The contracts lack validations against address(0) for crucial parameters such as state variables or interfaces of other contracts, especially if they are immutable or can’t be updated later through other functions. That will decrease the risk of human error.

3. Magic numbers.

TokenVesting.sol: addRecepients() -> 100;

The contracts use magic numbers without explanations or descriptive variable names. Magic numbers can make the code difficult to read, understand, and maintain, as the purpose of these numbers is not immediately clear.

Replace magic numbers with named constants or variables, and provide explanations or comments to clarify their purpose.

4. Custom errors.

Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Require checks are rather expensive, especially when it comes to deployment cost, and it is difficult to use dynamic information in them.

5. Unused check.

TokenVesting: constructor();

Token Vesting contract inherits an Ownable contract from OZ which has a prebuilt function to prevent setting the owner as zero address, thus checking for zero address for the owner is no longer needed and unreachable.

Remove the check for zero address for the owner in the constructor.

6. Solidity version not fixed (and not the latest).

The Solidity version in the contracts is not fixed but instead uses the ^ (caret) symbol, which allows the compiler to use any version greater than or equal to the specified version, up to the next major version.

While this approach can provide flexibility, it can also lead to unexpected behavior or vulnerabilities if a newer, untested version of the compiler is used. For example, if a breaking change or a bug is introduced in a later compiler version, it could potentially affect the contract's functionality or security.

Also, the project setup expects the solc 0.8.21, while it is recommended to use the latest stable version - which is currently 0.8.28

It is recommended to fix the Solidity version to a specific, stable version that has been thoroughly tested with the contract. Specifically, it is advisable to use the latest stable version available at the time of development to benefit from the most recent security patches and optimizations.

7. Unused interface

IAUT.sol is not used in the system and can be safely removed - especially since it is identical to regular IERC20 interface

8. Missing events.

Distributor.sol: distribute();

TokenVesting.sol: _createVestingSchedule(), revoke(), withdraw(), release().

Certain state-changing functions in the contracts do not emit events to log the changes. Events are crucial for tracking state changes on the blockchain, providing transparency, and allowing off-chain applications to respond to these changes.

Add events to log state changes.

9. Confusing naming

TokenVesting.sol, VestingSchedule.cliff

During the creation of the vesting schedule in createVestingSchedule(), the function receives the ”cliff” parameter which corresponds to the cliff duration. However it is modified to be stored in the schedule structure as a cliff end timestamp. Therefore it creates ambiguity in the naming of function parameter and storage field. Considered renaming either the storage field (e.g. cliffEnd) or function parameter (e.g. cliffDuration).

10. nonReenterant is not required

TokenVesting.sol. The nonReentrant modifier is not required in the contract and can be safely removed:

- addRecepients(), withdraw() - no external calls except token balance and transfer, which refers to Aut token (ERC20 compatible) controlled by admin;
- release() - no external calls except token transfer, which refers to Aut token (ERC20 compatible).

It can be safely removed to save some gas during calls.

11. Duplicating viewer

TokenVesting.getToken() duplicates the public variable _token.

12. Unnecessary result handling

Distributor.sol: sendTokens();

“success” variable is unnecessary as a whole error handling, as for the Aut and c-aut tokens both burn() and transfer() always return true, thus the error will never be reached.

RECOMMENDATION:

Consider correcting all listed issues to increase code readability and logic optimization.

POST-AUDIT:

Resolved points: 1, 2, 3, 5, 7, 8, 9, 10, 11, 12

Unaddressed but verified:
6) Solidity Version Not Fixed - After internal discussions, the team has set a fixed version 0.8.20 for contracts and a floating version ^0.8.0 for interfaces;

4) Custom Errors - Following internal discussions, the team decided not to replace the existing require statements with custom errors at this time.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

Aut

Constructor

- ✓ Returns `Aut Labs` when getting the token name
- ✓ Returns `AUT` when getting the token symbol
- ✓ Returns `18` when getting the decimals
- ✓ Returns `100000000000000000000000000` when getting the total supply

Approve

- ✓ Allows to approve spending (218ms)
- ✓ Allows to approve spending with zero balance (485ms)
- ✓ Reverts if spender is zero
- ✓ Allows to transfer without approval (218ms)
- ✓ Reverts if approver is zero address (225ms)

Transfer

- ✓ Allows to transfer tokens (40ms)
- ✓ Reverts when trying to transfer tokens to zero address
- ✓ Reverts when user doesn't have enough balance
- ✓ Reverts when spender is zero address

TransferFrom

- ✓ Allows to transfer tokens from user3 to user2 (46ms)
- ✓ Reverts when user doesn't have enough allowance
- ✓ Reverts when user doesn't have enough balance

Distributor

Constructor

- ✓ Sets all values properly (51ms)
- ✓ Should revert if one of the parameters is invalid (1894ms)

Distribute

- ✓ Should allow to distribute tokens (1686ms)
- ✓ Should revert if caller is not owner
- ✓ Should revert if balance of aut on distrubutor contract is less than 100 mil

Investors Vesting

Constructor

- ✓ Should sets all values properly
- ✓ Should revert if token is invalid
- ✓ Should revert if owner is zero address (45ms)

Early Contributors Vesting

Constructor

- ✓ Should sets all values properly
- ✓ Should revert if token is invalid
- ✓ Should revert if owner is zero address

Advisors Vesting

Constructor

- ✓ Should sets all values properly
- ✓ Should revert if token is invalid
- ✓ Should revert if owner is zero address

Token Vesting

Constructor

- ✓ Sets all values properly (51ms)
- ✓ Should revert if token is invalid (219ms)
- ✓ Should revert if owner is zero address
- ✓ Should revert if duration is zero
- ✓ Should revert if release interval is zero

Vesting creation

- ✓ Should allow to create vesting (1852ms)
- ✓ Should revert if beneficiary is zero address
- ✓ Should not allow to create vestings in sum more than withdrawable amount
- ✓ Should revert if tokens are not enough to create vesting
- ✓ Should revert if amount is zero
- ✓ Should revert if duration is less than cliff

Add recipients

- ✓ Should allow to add recipients (2957ms)
- ✓ Should revert if allocations in total is more than 100
- ✓ Should revert of arrays length is not equal
- ✓ Should revert if array length is bigger than the maximum allowed (735ms)

Revoke

- ✓ Should allow to revoke the user
- ✓ Should revert if vesting was already revoked
- ✓ Should revert if vesting is not revocable (2525ms)
- ✓ Should send releasable tokens to the user and revoke then (236ms)

Withdraw

- ✓ Should allow to withdraw tokens
- ✓ Should revert if not enough tokens to withdraw

Release

- ✓ Should allow to release tokens (73ms)
- ✓ Should revert if vesting is revoked
- ✓ Should revert when release from not beneficiary or owner

Access control

- ✓ Should revert if caller is not owner

Getters

- ✓ Should allow to get vesting id at index (52ms)
- ✓ Should give vesting schedule information for a given holder and index (201ms)

Additionally Blaize team developed a set of scenarios to test edge cases of the functionality and performed a round of exploratory testing.
Note: team also developed tests for cAut and Reputation Mining contracts, though these tests will be included into the next audit iteration.

RESULTING COVERAGE

FILE	% STMTS	% BRANCH	% FUNCS
Investors.sol	100	100	100
Team.sol	100	100	100
TokenVesting.sol	100	100	100
UtilsRegistry.sol	100	100	100
Distributor.sol	100	100	100
AUT.sol	100	100	100
cAUT.sol	100	100	100

NATIVE TESTS OVERVIEW

The Āut Labs team supplied native tests for the audited contracts. However, the test coverage was partial, leaving certain areas of the provided contracts untested.

Disclaimer

The information presented in this report is an intellectual property of the customer, including all the 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's 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.