CPRO Vesting Audit Report

Document version: **1.1.1**

Auditors:
Srdjan Delić
Tatjana Radovanović

Audit date:
27.10.2025

Contents:

Overview

Repository / commit hash / tag: ______________________
Files / folders included in the audit: locking.sol
Language / compiler version: Solidity ^0.8.28
Testing networks: Remix, Sepolia, ChatGPT
Audit start date: 2025-10-27
Audit end date: 2025-10-27

About CPRO Vesting

CPRO (CryptoProccessor) Vesting is an ownable smart contract developed by the CPRO Team for the Ethereum chain. The main purpose of the vesting smart contract is to release CPRO tokens based on predefined scheduling intervals. The tokens are locked and are released gradually over time. This benefits transparency and increases trust in the owner and token itself by removing any manual intervention and focusing on automated delivery.

Vesting cliffs, schedules, and milestones are stored on the blockchain making them clear and unchangeable.

Disclaimer

The audit team will implement extensive and exhaustive internal tooling and methodologies into providing a complete token audit process. The audit team **cannot** guarantee that all potential vulnerabilities or risks will be identified and as such holds **no responsibility** if issues are later identified in any further step of the development or deployment process for the token smart contract(s) being audited.

The audit is focused on Solidity contracts security and quality of code with the consideration for the implementation and adherence of the best, industry-wide practices implemented in the said smart contracts.

In case of future smart contact revisions or edits, all previous audits will be deemed as unverified/deprecated and **invalidated**. For even minor edits to the audited contracts, the same audit methodologies should be applied again in order to achieve the same level of confidence and security.

Risk Classification

Severity chart:

- **Critical** - High impact and high likelihood of happening. Requires immediate attention and fixing
- **High** - Medium impact and high likelihood of happening. Requires attention and fixing
- **Medium** - Medium impacts and medium likelihood of happening. No urgent attention required, but recommended addressing before next deployment/testing cycle
- **Low** - Low impact and low likelihood of happening. Can be postponed to the next development cycle
- **Informative** - Audit team's own suggestions for code,structure and security improvements and potential vulnerabilities that could manifest in the near future, but waren't part of the required token specifications

Tools

Slither - Hardhat/Foundry projects
Aderyn - Hardhat/Foundry projects
AI Tools - ChatGPT, Claude, RemixAI A ssistant

Protocol Summary

Description

Documentation

The CPRO project documentation is under an NDA and is not available for public view.

Actors and Roles

Actors

- Token_Team: Token smart contract developers and core team members.
- Users: Receivers of the CPRO token via scheduling contracts.

Key Components

CPROVesting.sol is the vesting smart contract developed by the CPRO Team and it manages the gradual release of ERC20 CPRO tokens.

This contract implements a token vesting system where tokens are locked and released gradually to beneficiaries according to predefined schedules.

Basic features:

- Linear vesting with cliff periods
- Owner-managed vesting schedules
- Revocable vesting by owner
- Multiple beneficiaries support
- Reentrancy protection

Audit Scope

- CPROVesting.sol

Result Summary

There are some security concerns and some critical issues found during the audit and a handful of high/medium robustness issues around ERC-20 safety and revocation behavior.

The smart contract has security features:

- Uses ReentrancyGuard for claimTokens()
- Uses Ownable for access control
- Uses SafeERC20 (implicitly via OZ's IERC20.transfer/transferFrom in ^0.8.28)
- Prevents reentrancy, overflows (Solidity ^0.8.x), and front-running (via nonReentrant)

The following findings are a consolidated set of issues and observations from various tools and audit engineers.

Findings

1. **Revocation permanently blocks already-vested but unclaimed tokens**
   Severity: CRITICAL
   Location: CPROVesting.sol, function revokeVesting(address beneficiary)
   Description: revokeVesting() sets revoked = true, transfers unvested to owner, but does not pay the beneficiary their already-vested (but unclaimed) portion nor leave a way to claim it. Vested tokens can become irrecoverably stuck in the contract after revocation (beneficiary loses funds, protocol state inconsistent with expected vesting semantics)

   Recommendation:

   Calculate vested = _calculateVestedAmount(schedule).
   Compute dueToBeneficiary = vested - schedule.claimedAmount.
   Pay dueToBeneficiary to the beneficiary.
   Return only the unvested remainder to the owner.
   Then set revoked = true

   CPRO Team: addressed
   Audit Team: Waiting for fix implementation

2. **Unsafe ERC-20 interactions (non-standard tokens / fee-on-transfer)**
   Severity: HIGH
   Location: CPROVesting.sol, functions createVestingSchedule (transferFrom), claimTokens (transfer), revokeVesting (transfer), emergencyWithdraw (transfer)
   Description: The contract relies on IERC20.transfer/transferFrom returning bool and use require(...). This presumes that the CPROToken by default will return bool success or have burn mechanics, which we shouldn't presume and it's better to have safe-guards in place.

   Recommendation:

   Use OpenZeppelin SafeERC20 everywhere and do not trust the returned bool. Also handle fee-on-transfer by updating accounting based on actual transferred amount if you want to support such tokens

   CPRO Team: addressed
   Audit Team: Waiting for fix implementation

3. **emergencyWithdraw ignores transfer return value**
   Severity: HIGH
   Location: CPROVesting.sol, function emergencyWithdraw()
   Description: IERC20(_token).transfer(owner(), amount) call result ignored

   Recommendation:

   Use safeTransfer

   CPRO Team: addressed
   Audit Team: Waiting for fix implementation

4. **Token transfer fees may apply when claiming tokens**
   Severity: MEDIUM
   Location: CPROVesting.sol, function claimTokens()
   Description: claimTokens() increases claimedAmount by claimableAmount before transfer. If the token takes a fee, beneficiary receives less than claimableAmount but claimedAmount still advances by the full amount, preventing later recovery.

   Recommendation:

   Document that the vesting token must be a valid ERC20 token without any fees on transfers or burns.

   CPRO Team: in_process
   Audit Team: Waiting for fix implementation

5. **revokeVesting not marked nonReentrant**
   Severity: MEDIUM
   Location: CPROVesting.sol, function revokeVesting()
   Description: Adding a more robust way for preventing malicious tokens.

   Recommendation:

   Mark the function as nonReentrant

   CPRO Team: addressed
   Audit Team: Waiting for fix implementation

6. **Schedule lifecycle constraints**
   Severity: MEDIUM
   Location: CPROVesting.sol, function revokeVesting()
   Description: Once revoked, *exists* stays true. You cannot create a new schedule for the same beneficiary

   Recommendation:

   On revoke, set exists = false after settling balances

   CPRO Team: addressed
   Audit Team: Waiting for fix implementation

7. **Gas optimization**
   Severity: MEDIUM
   Location: CPROVesting.sol, beneficiaries array
   Description: beneficiaries array grows indefinitely (no cleanup after revocation)

   Recommendation:

   Consider removing revoked beneficiaries to save gas on iteration (if needed later)

   CPRO Team: acknowledged
   Audit Team: Waiting for fix implementation

Security Controls Review

Control	Status	Notes
Access Control	✅	Owner-only for schedule creation and revocation
Reentrancy Protection	✅	Properly implemented via `nonReentrant`
Token Transfer Validation	✅	Uses return-value checks on all ERC20 transfers
Vesting Schedule Immutability	✅	Once created, schedules cannot be overwritten
Revocation Logic	✅	Returns unvested tokens to owner securely
Timestamp Logic	✅	Proper cliff and vesting time validation
Beneficiary Isolation	✅	Each user’s schedule is independent

Emergency Withdraw	✅	Restricted and prevents withdrawal of vesting token
Upgradeable	❌	Static implementation (non-upgradeable by design)

Gas & Performance Observations

- Efficient per-user mapping, but could grow linearly with beneficiary count.
- Linear vesting calculation uses basic arithmetic — gas-efficient.
- Beneficiary list iteration is not implemented (avoids scalability issues).
- No unbounded loops in external functions.