# HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: Astra
**Date**:     August 29th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

## Document

| | |
|---|---|
| **Name** | Smart Contract Code Review and Security Analysis Report for Astra. |
| **Approved By** | Evgeniy Bezuglyi | SC Department Head at Hacken OU |
| **Type** | Vesting |
| **Platform** | EVM |
| **Language** | Solidity |
| **Methods** | Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review |
| **Website** | https://astradao.org/ |
| **Timeline** | 18.04.2022 - 29.08.2022 |
| **Changelog** | 22.04.2022 - Initial Review<br>10.06.2022 - Second Review<br>29.08.2022 - Third Review |

www.hacken.io

2

# Table of contents

## Introduction

Hacken OÜ (Consultant) was contracted by Astra (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

## Scope

The scope of the project is smart contracts in the repository:

**Initial review scope**
**Repository:**
     https://github.com/astradao/astra-private
**Commit:**
     094078482cb6671d3610426edd4167c6c621985b
**Technical Documentation:** Yes
**JS tests:** Yes
**Contract:**
     File: ./astra-smartcontracts/main/version-6/treasury-vesting.sol
     SHA3: c1434c1951b60915205a52c70504c298202e8846968e0f22d4beee54

**Second review scope**
**Repository:**
     https://github.com/astradao/astra-private
**Commit:**
     5e8676d1534685929fec2923bb9de49f9a488236
**Technical Documentation:** Yes
**JS tests:** Yes
**Contract:**
     File: ./astra-smartcontracts/main/version-6/treasury-vesting.sol
     SHA3: 5ccc743dce353089ced1862b0c4104feda52507c480f5800de49990e

**Third review scope**
**Repository:**
     https://github.com/astradao/astra-private
**Commit:**
     5e8676d1534685929fec2923bb9de49f9a488236
**Technical Documentation:** Yes
**JS tests:** Yes
**Deployed Contracts Addresses:**
     Proxy: https://etherscan.io/address/0xaA2Da8F75A6F7C37dd2A3EdcCBF7897759C9C119#code
     Implementation: https://etherscan.io/address/0x5b4d61760fae5fce7f362a600e038207fab1de5f#code
**Contract:**
     File: ./astra-smartcontracts/main/version-6/treasury-vesting.sol
     SHA3: 5ccc743dce353089ced1862b0c4104feda52507c480f5800de49990e

## Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions |
| Medium | Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations. |
| Low | Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution |

www.hacken.io

## Executive Summary

The Score measurements details can be found in the corresponding section of the [methodology](methodology).

### Documentation quality

The Customer provided appropriate functional and technical requirements. The total Documentation Quality score is **10** out of **10**.

### Code quality

The total CodeQuality score is **10** out of **10**. Code is easy to read.

### Architecture quality

The architecture quality score is **10** out of **10**. The structure is clear.

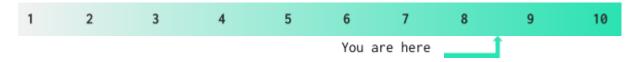### Security score

As a result of the audit, security engineers found **2** medium and **1** low severity issues. The security score is **8** out of **10**.

All found issues are displayed in the "Findings" section.

### Summary

According to the assessment, the Customer's smart contract has the following score: **8.6**.

## Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

| Item | Type | Description | Status |
|------|------|-------------|--------|
| **Default Visibility** | SWC-100 SWC-108 | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | Passed |
| **Integer Overflow and Underflow** | SWC-101 | If unchecked math is used, all math operations should be safe from overflows and underflows. | Passed |
| **Outdated Compiler Version** | SWC-102 | It is recommended to use a recent version of the Solidity compiler. | Failed |
| **Floating Pragma** | SWC-103 | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | Passed |
| **Unchecked Call Return Value** | SWC-104 | The return value of a message call should be checked. | Not Relevant |
| **Access Control & Authorization** | CWE-284 | Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users. | Passed |
| **SELFDESTRUCT Instruction** | SWC-106 | The contract should not be destroyed until it has funds belonging to users. | Not Relevant |
| **Check-Effect-Interaction** | SWC-107 | Check-Effect-Interaction pattern should be followed if the code performs ANY external call. | Failed |
| **Uninitialized Storage Pointer** | SWC-109 | Storage type should be set explicitly if the compiler version is < 0.5.0. | Not Relevant |
| **Assert Violation** | SWC-110 | Properly functioning code should never reach a failing assert statement. | Not Relevant |
| **Deprecated Solidity Functions** | SWC-111 | Deprecated built-in functions should never be used. | Not Relevant |
| **Delegatecall to Untrusted Callee** | SWC-112 | Delegatecalls should only be allowed to trusted addresses. | Not Relevant |
| **DoS (Denial of Service)** | SWC-113 SWC-128 | Execution of the code should never be blocked by a specific contract state unless it is required. | Passed |
| **Race** | SWC-114 | Race Conditions and Transactions Order | Passed |

| Conditions | | Dependency should not be possible. | |
|---|---|---|---|
| Authorization through tx.origin | SWC-115 | tx.origin should not be used for authorization. | Not Relevant |
| Block values as a proxy for time | SWC-116 | Block numbers should not be used for time calculations. | Passed |
| Signature Unique Id | SWC-117 SWC-121 SWC-122 | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. | Not Relevant |
| Shadowing State Variable | SWC-119 | State variables should not be shadowed. | Passed |
| Weak Sources of Randomness | SWC-120 | Random values should never be generated from Chain Attributes. | Not Relevant |
| Incorrect Inheritance Order | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. | Passed |
| Calls Only to Trusted Addresses | EEA-Level-2 SWC-126 | All external calls should be performed only to trusted addresses. | Passed |
| Presence of unused variables | SWC-131 | The code should not contain unused variables if this is not justified by design. | Passed |
| EIP standards violation | EIP | EIP standards should not be violated. | Not Relevant |
| Assets integrity | Custom | Funds are protected and cannot be withdrawn without proper permissions. | Passed |
| User Balances manipulation | Custom | Contract owners or any other third party should not be able to access funds belonging to users. | Not Relevant |
| Data Consistency | Custom | Smart contract data should be consistent all over the data flow. | Passed |
| Flashloan Attack | Custom | When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. | Not Relevant |
| Token Supply manipulation | Custom | Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer. | Not Relevant |
| Gas Limit and Loops | Custom | Transaction execution costs should not depend dramatically on the amount of | Passed |

| | | data stored on the contract. There should not be any cases when execution fails due to the block gas limit. | |
|---|---|---|---|
| **Style guide violation** | **Custom** | Style guides and best practices should be followed. | Passed |
| **Requirements Compliance** | **Custom** | The code should be compliant with the requirements provided by the Customer. | Passed |
| **Repository Consistency** | **Custom** | The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | Passed |
| **Tests Coverage** | **Custom** | The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested. | Passed |

www.hacken.io

## System Overview

Astra is a mixed-purpose system that includes the contract from the audit scope:

- TokenVesting — simple vesting contract with the ability to connect Chainlink Keeper for automated vesting releasing. Airdrop function for p2p giveaways is provided.

## Privileged roles

- The owner has the ability to:
  - create vestings
  - revoke vestings if a revocable parameter was provided on creation
  - withdraw unvested assets
  - update period for Chainlink Keeper automated releasing

## Some keypoints for user

- The owner can revoke a vesting if, on creation, such a parameter was provided. On revoking, all vested tokens till the moment are automatically released to the beneficiary account.

# Findings

## ■■■■ Critical

No critical severity issues were found.

## ■■■ High

No high severity issues were found.

## ■■ Medium

### 1. Potential Gas limit exceeding

A loop around an only increasing array may exceed Gas in the future when the array becomes too big.

This could lead to the breaking of the automated tool.

**Contract**: treasury-vesting.sol

**Function**: performUpkeep

**Recommendation**: implement an array of only actual vesting ids, perform automation only for those ones.

**Status**: Fixed (second scope)

### 2. Unchecked integer overflow/underflow

Unchecked math is a common cause of losing assets and double-spending attacks. This particular case may cause such problems if the provided *token* violates *ERC-20* standard.

Leaving math unchecked may become dangerous, along with other errors. It may cause problems in future development.

**Contract**: treasury-vesting.sol

**Functions**: multisendToken, getLastVestingScheduleForHolder

**Recommendation**: provide corresponding checks.

**Status**: Fixed (second scope)

### 3. Possible locking of Ether

Empty receive and fallback functions may cause locking of *Ether* on the smart contract without any ability to transfer it to another account.

Leaving payable receive and fallback functions on a contract that does nothing with *Ether* may lead to unpleasant results for users who accidentally call the contract with attached *Ether*.

**Contract**: treasury-vesting.sol

**Functions**: receive, fallback

**Recommendation**: remove these functions.

**Status**: Fixed (second scope)

### 4. Check-Effect-Interaction pattern violation

This pattern is designed to prevent reentering attacks, but it saves Gas and keeps the project's code clear.

Violation of the Check-Effect-Interaction pattern may cause problems in future development.

**Contract**: treasury-vesting.sol

**Function**: revoke

**Recommendation**: put any external calls or calls that make those ones inside at the end of the function.

**Status**: Reported

### 5. Chainlink automation may be prevented

A user may force call *performUpkeep* function to prevent next automated upkeep.

It may happen because the *keeperLastUpdatedTime* variable is updated even if the upkeep is not done. The variable is used in the validation of the upkeep applicable, so automated upkeep may never happen.

**Contract**: treasury-vesting.sol

**Function**: performUpkeep

**Recommendation**: update the variable only when upkeep is applicable and done.

**Status**: New

## ■ Low

### 1. Unused events

Events *Released* and *Revoked* are defined but never used.

**Contract**: treasury-vesting.sol

**Function**:

**Recommendation**: if mentioned events should be emitted, do this or remove them.

**Status**: Fixed (second scope)

### 2. Unused modifier

Modifier *onlyIfVestingScheduleExists* is defined but never used.

**Contract**: treasury-vesting.sol

**Recommendation**: if mentioned modifier should be used, do this or remove it.

**Status**: Fixed (second scope)

### 3. Boolean equality

Boolean constants can be used directly and do not need to be compared to *true* or *false*.

**Contract**: treasury-vesting.sol

**Functions**: performUpkeep, _computeReleasableAmount, revoke, onlyIfVestingScheduleExists, onlyIfVestingScheduleNotRevoked

**Recommendation**: remove the equality to the boolean constant.

**Status**: Fixed (second scope)

### 4. Functions that can be declared as external

To save Gas, public functions that are never called in the contract should be declared as *external*.

**Contract**: treasury-vesting.sol

**Functions**: addUserDetails, revoke, withdraw, getWithdrawableAmount, computeNextVestingScheduleIdForHolder, getLastVestingScheduleForHolder

**Recommendation**: the functions above should be declared as *external*.

**Status**: Fixed (second scope)

### 5. Copy-pasting of well-known contracts

It is better to import well-known contracts from the initial source, for example, from the OpenZeppelin repository. These contracts are in development, so importing them from open libraries will make code more flexible.

**Contract**: treasury-vesting.sol

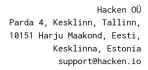**Imports**: SafeMath, IERC20, ERC20, SafeERC20, Ownable (OwnableUpgradeable), Context, Initializable

**Recommendation**: change local imports to imports from the initial source.

**Status**: Fixed (second scope)

### 6. Mixing levels of abstraction

Several *view* functions create abstraction around contract data defined, but they are not used in the contract code or used particularly.

**Contract**: treasury-vesting.sol

**Functions**:        getVestingSchedulesCount,        getVestingSchedule,
getCurrentTime

**Recommendation**: provide abstraction levels consciously.

**Status**: Fixed (second scope)

### 7. Outdated compiler version

Using an outdated compiler version can be problematic, especially if
publicly disclosed bugs and issues affect the current compiler
version.

**Contract**: treasury-vesting.sol

**Recommendation**: use a recent version of the Solidity compiler.

**Status**: Reported

# Disclaimers

## Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

## Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.

www.hacken.io