HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Astra
Date: June 10th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Astra
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://astra.finance
Timeline	02.05.2022 - 10.06.2022
Changelog	06.05.2022 - Initial Review 03.06.2022 - Second Review 10.06.2022 - Third Review

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:
    309f76a
Technical Documentation:
    Type: Whitepaper
    Link: Public Facing ASTRA Whitepaper

    Type: Public documentation
    Link: https://docs.astradao.org/

Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:

```
File: ./astra-smartcontracts/main/version-6/astr.sol
SHA3: 7c29947d104a46fce101700c039f1f0d7eb144c4b1f1ecc549835d4c93eda617

File: ./astra-smartcontracts/main/version-6/lm-pool-erc721.sol
SHA3: 8d093055dc692792fa384bc0657017a98f3754aafea10b77238a48ee18d09f0d

File: ./astra-smartcontracts/main/version-6/upgrade/ERC20UpgradeSafe.sol
SHA3: ba7df049f29449ff84f72138a34582fe010ff774a81d2987bef1fa084ddb9354
```

Second review scope

Repository:
    https://github.com/astradao/astra-private
Commit:
    a2bbe59
Technical Documentation:
    Type: Whitepaper
    Link: Public Facing ASTRA Whitepaper

    Type: Functional and technical requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses: No
Contracts:

```
File: ./astra-smartcontracts/main/version-6/astr.sol
SHA3: 37f4508bf220011de4791432f31ca91b80b67c9bc6c62897839644f11a8fab7b

File: ./astra-smartcontracts/main/version-6/lm-pool-erc721.sol
SHA3: 686ea61ace5b2512a3520750231e0ec71ef4556d969c4f7602ddb46f5cd66a91
```

Third review scope

Repository:
https://github.com/astradao/astra-private
Commit:
5e8676
Technical Documentation:
Type: Whitepaper
Link: Public Facing ASTRA Whitepaper

Type: Functional and technical requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses: No
Contracts:
File: ./astra-smartcontracts/main/version-6/astr.sol
SHA3: f82e0da1dcbd68bffb5b8c14f5d8a3f3af66a878dc3704c6751baf6a12bfe088

File: ./astra-smartcontracts/main/version-6/lm-pool-erc721.sol
SHA3: ca496445a106c33fc865a5efb6a302f4d36adbd3fec99ecedde6aa9d1578f551

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.

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

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

Code quality

The total CodeQuality score is **9** out of **10**. Unit tests were provided, and the official Solidity code style was followed. Files naming convention and functions order rules are violated.

Architecture quality

The architecture quality score is **8** out of **10**.

Security score

As a result of the audit, security engineers found **2** low severity issues. The security score is **10** 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: **9.7**.

1
2
3
4
5
6
7
8
9
10
The final score

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.	Not Relevant
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.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Not Relevant
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.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
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 Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
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.	Passed
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.	Passed
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.	Passed
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.	Passed
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed

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
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

ASTRA DAO is a complex system using different investment strategies with the following contracts:

- *Token* – simple ERC-20 token that mints all initial supply to the specified contract. However, additional minting is allowed.
- *LmPoolV3* – a contract that rewards users with ASTRA tokens for deposit of their ERC721 tokens.

Privileged roles

- The owner can:
  - add a minter address.
  - add new tokens to the pool.
  - add new lp to the pool.
  - add new vault period.
  - set chef contract address in which users can eventually stake their rewards.
  - withdraw all tokens from the Liquidity Pool contract.
  - distribute rewards between users.
- The initializer can:
  - initialize a contract.

Findings

■■■■ Critical

1. **Access violation.**

   The function is available for calling by everyone.

   Any amount of tokens can be distributed between users in the pool, and users can withdraw them. Eventually, the ERC20 Astra contract will be drained.

   Contract: lm-pool-erc721.sol

   Function: distributeExitFeeShare

   Recommendation: Validate ownership.

   Status: Fixed (Revised commit: a2bbe5)

2. **Missing file.**

   Missing import file *IERC721Receiver.sol*, line 12.

   Therefore contract cannot be compiled.

   Contracts: lm-pool-erc721.sol

   Function: -

   Recommendation: Add the missing file.

   Status: Fixed (Revised commit: a2bbe5)

■■■ High

1. **Token minting.**

   According to the tokenomics, maximum total supply is 100,000,000,000,000 (100 trillion), but the functionality allows the owner to mint more.

   Contracts: astr.sol

   Function: mint

   Recommendation: Remove the ability to mint more than stated in tokenomics.

   Status: Mitigated (The whitepaper has been updated)

2. **Minter cannot be deleted.**

   The functionality allows adding a minter address. However, there is no facility to delete it.

   Therefore minter can not be revoked if necessary.

   Contracts: ERC20UpgradeSafe.sol

   Function: mintNewTokens

Recommendation: Allow to revoke minters.

Status: Fixed (Revised commit: a2bbe5)

3. **The owner can withdraw all reward tokens.**

   The owner can withdraw all tokens from the contract from the LmPool contract.

   This may affect users' reward funds.

   Contracts: lm-pool-erc721.sol

   Function: emergencyWithdrawASTR

   Recommendation: Remove the ability of the owner to withdraw user rewards or update the documentation accordingly.

   Status: Fixed (Revised commit: 5e8676)

4. **Pausing all token transfers.**

   The functionality allows the owner to pause all the token transfers anytime. Pausing functionality should be limited by clear contract rules. The documentation does not mention the functionality of transfers stopping.

   Contracts: astr.sol

   Function: pause

   Recommendation: Remove pausing functionality or update the documentation accordingly.

   Status: Mitigated (The whitepaper has been updated)

5. **Potential DoS.**

   The function iterates over all users in the specified pool.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of users is large enough.

   Contracts:lm-pool-erc721.sol

   Function: viewRewardInfo, updateBlockReward

   Recommendation: Do not iterate over all users.

   Status: Fixed (Revised commit: a2bbe5)

6. **Potential DoS.**

   The function iterates over all pools and their users. Changes the state every iteration.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of pools, users is large enough.

   Contracts:lm-pool-erc721.sol

   Function: distributeFlatReward

Recommendation: Do not iterate over all pools and users.

Status: Fixed (Revised commit: 5e8676)

7. **Potential DoS.**

The function iterates over all users in the specified pool. Changes the state every iteration.

Gas consumption can differ a lot between different transactions. Possible DoS if the number of pools, users is large enough.

Contracts:lm-pool-erc721.sol

Function: distributeIndividualReward

Recommendation: Do not iterate over all users.

Status: Fixed (Revised commit: a2bbe5)

🟩🟩 Medium

1. **Property never used.**

Property *rewardDebt* declared for the *UserInfo* struct, mentioned in the documentation, but never used.

Contract: lm-pool-erc721.sol

Functions: global declaration

Recommendation: Review and fix the logic.

Status: Fixed (Revised commit: 5e8676)

2. **Dev address is unused.**

Variable *devaddr* declared, established, but never used. Should transfer ownership instantly to *devaddr* address upon contract initialization like in the previous scope?

Contract: lm-pool-erc721.sol

Functions: global declaration

Recommendation: Review and fix the logic.

Status: Fixed (Revised commit: 5e8676)

🟦 Low

1. **Floating pragma.**

The contracts use floating pragma ^0.6.0, ^0.6.12.

Contract: ERC20UpgradeSafe.sol, lm-pool-erc721.sol, astr.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised commit: a2bbe5)

2. **Outdated Compiler Version.**

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

   Contract: lm-pool-erc721.sol, astr.sol

   Recommendation: Use a recent version of the Solidity compiler.

   Status: Reported

3. **Unused functions.**

   _setupDecimals is defined but never used.

   Contract: ERC20UpgradeSafe.sol

   Function: _setupDecimals

   Recommendation: Remove this function or make the contract abstract.

   Status: Fixed (Revised commit: a2bbe5)

4. **Unused variables.**

   ABP, highestStakerInPool, totalAllocPoint, timelock, coolDownStart, are defined but never used.

   Contract: lm-pool-erc721.sol

   Function: -

   Recommendation: Remove these variables.

   Status: Fixed (Revised commit: a2bbe5)

5. **State variables that can be declared constant.**

   To save Gas, constant state variable *dayseconds* should be declared *constant*.

   Contract: lm-pool-erc721.sol

   Function: initialize

   Recommendation: Add the *constant* attribute to state variables that never change.

   Status: Fixed (Revised commit: a2bbe5)

6. **Overwhelmed code.**

   Unneeded reassignment is provided. *dayseconds* variable could be defined already with actual value.

   Contract: lm-pool-erc721.sol

   Function: initialize

Recommendation: Update the code and use assignment in place of declaration.

Status: Fixed (Revised commit: a2bbe5)

7. **No events on state variables changings.**

   It is recommended to emit events on important state changes.

   Contracts: lm-pool-erc721.sol

   Functions: add, checkEligibleAmount, withdrawASTRReward

   Recommendation: Emit events on important state changes.

   Status: Fixed (Revised commit: a2bbe5)

8. **Unused imports.**

   Imported ./common/ERC20.sol, ./common/EnumerableSet.sol, ./common/Context.sol are not used.

   Contracts: lm-pool-erc721.sol

   Functions: -

   Recommendation: Remove unused imports.

   Status: Fixed (Revised commit: a2bbe5)

9. **Confused variable name.**

   startTime variable is supposed to be *block.number*, not timestamp.

   This may confuse whoever reads the contract.

   Contracts: astr.sol

   Function: -

   Recommendation: Fix name appropriately.

   Status: Fixed (Revised commit: a2bbe5)

10. **Redundant require statement.**

    The checking if *block.timestamp <= `cool down period plus 8 days`* is redundant because it has already been checked in the initial `if` statement.

    Contracts: lm-pool-erc721.sol

    Function: withdraw

    Recommendation: Remove the redundant require statement.

    Status: Fixed (Revised commit: a2bbe5)

11. **Documentation inconsistency.**

Function signature `*setTimeLockAddress(address)*` mentioned in the provided documentation but removed in the new audit scope.

Contracts: lm-pool-erc721.sol

Recommendation: Fix inconsistency.

Status: Reported

12. **Redundant modifier.**

The modifier *nonReentrant* is redundant because there are no circumstances for reentrancy attacks.

Contracts: lm-pool-erc721.sol

Functions: withdrawASTRReward, deposit

Recommendation: Remove the redundant modifier.

Status: Fixed (Revised commit: 5e8676)

13. **Unused import.**

Imported @openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol is not used.

Contracts: astr.sol

Recommendation: Remove unused import.

Status: Fixed (Revised commit: 5e8676)

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.