QuillAudits

Audit Report
May, 2024

For

ASTRA
DAO

Table of Content

Table of Content

Executive Summary

Project Name	Astra DAO LaunchPad
Overview	Astra DAO LaunchPad contains multiple contracts. With the factory contract, launchpads could be requested, approved and created. There is also a whitelist contract that integrates the PureFi KYC verification proof and an easy way for the contract owner to whitelist addresses. Launchpad could be linked to a vesting contract and enabled. From the launchpad, users can purchase tokens using the native token, supported stable coins, or other tokens that make swaps on the exchange contract. The Axelar General Message Passing was adopted in the crossChainSaleManager for the purpose of updating the staking info between evm compatible chains.
Timeline	23rd April 2024 to 20th May 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	This audit aimed to analyse the Astra DAO LaunchPad Codebase for quality, security, and correctness. 1. launchpad.sol 2. crossChainSaleManager.sol 3. astraDAOWhitelist.sol 4. launchpadConfiguration.sol 5. launchpadFactory.sol 6. launchpadVesting
Source Code	*https://github.com/Astra DAO LaunchPad/Astra DAO LaunchPad-contracts-audit/tree/main*
Commit Hash	391b73c12098a874ccf5ac6b677a6e273536533f
Branch	Main
Fixed In	*https://github.com/astradao/astradao-smart-contracts/commit/e95eff5164669e21f164fc44a9ac69f428a1486b4*
Commit Hash	43d131b75d9bcfddf1f4acec9a7b6a99c4e5150a

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	1	1	2

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four severity levels, each of which has been explained below.

High Severity Issues

A high severity issue or vulnerability means your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These issues were identified in the initial audit and successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Tokens are locked forever when the contract owner revokes a vesting schedule during the vesting period

Path

LaunchpadVesting.sol

Function

revoke()

```
function revoke(bytes32 vestingScheduleId⬆)
    external
    onlyOwner
    nonReentrant
    onlyIfVestingScheduleNotRevoked(vestingScheduleId⬆)
{
    VestingSchedule storage vestingSchedule = vestingSchedules[vestingScheduleId⬆];
    uint256 vestedAmount = _computeReleasableAmount(vestingSchedule);
    if (vestedAmount > 0) {
        //@audit are all of the tokens returned when vestingScheduleId is revoked?
        _release(vestingScheduleId⬆);

        uint256 unreleased = vestingSchedule.amountTotal.sub(vestingSchedule.released);
        vestingSchedulesTotalAmount = vestingSchedulesTotalAmount.sub(unreleased);
        vestingSchedule.revoked = true;

        bytes32 element = vestingSchedulesIds[vestingSchedulesIds.length.sub(1)];
        uint256 tempVestingId = userVestingScheduleId[vestingScheduleId⬆];
        vestingSchedulesIds[tempVestingId] = element;
        userVestingScheduleId[element] = tempVestingId;
        vestingSchedulesIds.pop();

        emit Revoked(vestingSchedule.beneficiary, vestingScheduleId⬆);
    }
}
```

Description

Users who purchase tokens from launchpad during sale period, with vesting contract enabled, will receive the initial release amount computed from the initial unlock percent and the rest sent into the vesting contract. This way, the user creates a vesting schedule. Users can afterwards invoke the release function during vesting periods or after vesting duration is over, to claim all of its tokens. The issue of locked tokens arises at a situation when the contract owner to the vesting contract revokes a vesting schedule id during vesting period.

When the revoke function is called during the vesting period, the _computeReleasable function computes how much token amount he is to receive. If users have a vestedAmount greater than zero, it will then call the _release function to send out the vestedAmount.

```
function _release(bytes32 vestingScheduleId↑) internal {
    VestingSchedule storage vestingSchedule = vestingSchedules[vestingScheduleId↑];
    uint256 vestedAmount = _computeReleasableAmount(vestingSchedule);
    require(vestedAmount > 0, "TokenVesting: no tokens are due");
    vestingSchedule.released = vestingSchedule.released.add(vestedAmount);
    address payable beneficiaryPayable = payable(vestingSchedule.beneficiary);
    vestingSchedulesTotalAmount = vestingSchedulesTotalAmount.sub(vestedAmount);
    _token.safeTransfer(beneficiaryPayable, vestedAmount);
    emit Released(vestingSchedule.beneficiary, vestedAmount);
}
```

The revoked vesting schedule becomes true and due to the onlyIfVestingScheduleNotRevoked added to the release public function. Users won't be able to collect their tokens. There will be left over tokens in the contract since not all purchased tokens are sent when revoked.

POC

https://gist.github.com/Ephraim-nonso/44fcfebe49f98b8399a67673bc1a8093

Recommendation

Design the contract to consider users whose vesting schedule has been revoked and ensure they can get all of their purchased tokens.

Status

Resolved

Astra Team Resolution

The vesting contract was redesigned to send out these left over tokens to the owner of the vesting contract, whenever there are revoked vesting schedule id during vesting period.

2. In crossChainSaleManager, when a removed chain is added back, this will hike the getWeightedAverageMultiplier value because this item will appear twice in the allChains array

Path

CrossChainSaleManager.sol

Function

removeCrossChainSaleManager

```
ftrace | funcSig
function setCrossChainSaleManager(string memory chain_↑, address crossChainSaleManagerAddress_↑) external onlyOwner
    require(crossChainSaleManagerAddress_↑ != address(0), "Zero address");
    //@audit if removed chain is to be added back, it will increase the number of appearance in allChains array
    //@audit This is because isChainAdded of that chain is falsy after it was removed.
    if (!isChainAdded[chain_↑]) {
        allChains.push(chain_↑);
        isChainAdded[chain_↑] = true;
    }
    crossChainSaleManagers[chain_↑] = crossChainSaleManagerAddress_↑;
}

//@audit-issue chain is not removed from allChains
ftrace | funcSig
function removeCrossChainSaleManager(string calldata chain_↑) external onlyOwner validChain(chain_↑) {
    delete crossChainSaleManagers[chain_↑];
    isChainAdded[chain_↑] = false;
}
```

Description

The contract owner can decide to remove a chain from the crossChainSlaeManager, and in the future, add this chain again. This will introduce an issue in how the getWeightedAverageMultiplier value will be computed.

The issue is as a result of the removeCrossChainSaleManager function failing to reduce the number of chains that make up the allChains array. Currently, when a chain is removed, it is deleted from the crossChainSaleManager mapping and set to false for isChainAdded. When this chain is added again, with the setCrossChainSaleManager function, it will push this chain again into the allChain array because the isChainAdded will be falsy.

This leaves an impact on the getWeightedAverageMultiplier function.

```
function getWeightedAverageMultiplier(address account_↑) external view returns (uint256) {
    if (allChains.length == 1) {
        uint256 nativeMultiplier;
        (, nativeMultiplier) = getNativeAmountAndMultiplier(account_↑);
        return nativeMultiplier;
    } else {
        uint256 totalAmount = 0;
        uint256 weightedMultiplierSum = 0;

        for (uint256 i = 0; i < allChains.length; i++) {
            ChainDetails storage details = crossChainStakingInfos[account_↑].details[allChains[i]];

            if (i == 0) {
                uint256 amount;
                uint256 multiplier;
                (amount, multiplier) = getNativeAmountAndMultiplier(account_↑);
                totalAmount += amount;
                weightedMultiplierSum += (amount * multiplier);
            } else {
                if (block.timestamp - details.lastUpdate > expirationTime) {
                    continue;
                }
                totalAmount += details.amount;
                weightedMultiplierSum += (details.amount * details.multiplier);
            }
        }

        if (totalAmount == 0) {
            return 10000000000000;
        }

        return weightedMultiplierSum.div(totalAmount);
    }
}
```

When computing the multiplier decimal, it will hike in value and coincidentally, on the launchpad contract, it will reflect on how the purchase limit of users are calculated.

```
ftrace | funcSig
function calculatePurchaseLimit(address _account↑) public view returns (uint256 limit) {
    address saleManager = ILaunchpadConfiguration(config).CROSS_CHAIN_SALE_MANAGER();
    uint256 multiplier = ICrossChainSaleManager(saleManager).getWeightedAverageMultiplier(_account↑);

    return baseAmount.mul(multiplier).div(MULTIPLIER_DECIMAL);
}
```

POC

```
function testSetandRemoveCrossChainSaleManager() external {
    vm.startPrank(owner);
    // a new crosschainsale manager is deployed for the matic chain
    address newCS = address(new CrossChainSaleManagerMock(owner, GATEWAY, GAS_RECEIVER, "Matic", CHEF_ADDRESS));
    crossChainContract.setCrossChainSaleManager("Matic", newCS);
    address get = crossChainContract.getCrossChainSaleManager("Matic");
    assertEq(get, newCS);

    // remove the chain and supposed to add back later in the future
    crossChainContract.removeCrossChainSaleManager("Matic");

    vm.warp(block.timestamp + 300);
    crossChainContract.setCrossChainSaleManager("Matic", newCS);


    // The "Matic" chain occupies both index 1 and 2 of the allChains array
    console.log(crossChainContract.getChainCount());
    console.log(crossChainContract.getChain(0)); // Matic
    console.log(crossChainContract.getChain(1)); // Matic
    vm.stopPrank();
}
```

Recommendation

The removeCrossChainSaleManager function should reduce the allChains array when removing the crossChainSaleManager of a particular chain.

Status

Resolved

Medium Severity Issues

1. LaunchPad End Time can be set in past

Path

LaunchPad.sol

Function

constructor

Description

The require statement in launchpad's constructor checks if

```
require(_saleStartTime < _saleEndTime, "Invalid sale time");
```

However, there is no check of these timestamps against the current **block.timestamp**. If a malicious request of launchpad gets approved where both **_saleEndTime** and **_saleStartTime** are set in the past then a launchpad contract will be deployed where **isSaleActive()** modifier will always revert thereby completely pausing **purchaseTokens()** function.

Recommendation

Consider checking if both start and end time are => block.timestamp

Status

Resolved

Low Severity Issues

1. Missing Zero Address Check

Path

LaunchpadVesting.sol

Function

setLaunchpad()

```
function setLaunchpad(address _launchpad) external onlyOwner {
    launchpad = _launchpad;
}
```

Description

The contract owner can set the launchpad address with this function but there are no adequate validation checks in it.

Recommendation

Add a check for the null address.

Status

Resolved

Informational Issues

1. Label state variables as immutable to avoid occupying storage slot

```
//@audit state variables set at the contract constructor level should be marked as immutable.
uint256 public START;
uint256 public CLIFF;
uint256 public DURATION;
uint256 public SLICE_PERIOD_SECONDS;
uint256 public INITIAL_UNLOCK;
```

Description

These state variables can not be changed after contract deployment but get set only once at the constructor level. Making these variables immutable will prevent creating more storage slots to store these values.

Recommendation

Make these variables immutable.

Status

Resolved

2. No length check for function parameters could cause a revert

Path

LaunchpadVesting.sol

Function

multisenToken()

```
/**
 * @dev Send tokens to multiple account
 */
function multisendToken(address[] memory recipients, uint256[] memory values) external {
    uint256 total = 0;
    for (uint256 i = 0; i < recipients.length; i++) {
        total = total.add(values[i]);
    }
    _token.safeTransferFrom(msg.sender, address(this), total);
    for (uint256 i = 0; i < recipients.length; i++) {
        _token.safeTransfer(recipients[i], values[i]);
    }
}
```

Description

If users pass in unequal parameters for the recipients and values, the function is certain to revert. In order to afford this revert, add a require check that validates that the length of both arrays are equal.

Recommendation

Check that recipients and values are of equal length.

Status

Acknowledged

Reference

https://solodit.xyz/issues/input-arrays-with-mismatched-length-will-make-addmanyusers-throw-openzeppelin-rndr-token-transfer-audit-markdown

3. Emit critical state changes

Path

LaunchpadVesting.sol

Modifier

onlyOwner

```
//@audit-issue failed to emit event for state changes
ftrace | funcSig
function updateBaseAmount(uint256 _baseAmount) external onlyOwner {
    baseAmount = _baseAmount;
}
```

Description

This variable is a critical state variable that serves as a determinant to compute other parameters, it'd be necessary to emit an event for this so it's easier to track anytime it's updated. Also there are multiple owner/admin only functions which are not emitting events.

Recommendation

Emit event for the update of baseAmout variable and other privileged setter functions.

Status

Resolved

Functional Tests Cases

Some of the tests performed are mentioned below:

- Should create a vesting schedule on purchasing tokens from launchpad
- Should allow the contract owner to revoke vesting schedule during vesting period
- Should check the balance of the vesting schedule after vesting is over with some vesting schedule revoked
- Should revert when multisendToken is triggered with unequal recipients and values array length
- Should remove cross chain sale manager and track the allChains array
- Should successfully approve requested launchpads with substantial amount of ERC20 approval

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Astra DAO LaunchPad codebase. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Astra DAO LaunchPad smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Astra DAO LaunchPad smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Astra DAO LaunchPad to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2024

For

ASTRA DAO

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com