HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Astrobot
Date: August 15th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Astrobot
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	ERC721 token;
Platform	EVM
Network	Ethereum,
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	-
Timeline	01.08.2022 - 15.08.2022
Changelog	05.08.2022 - Initial Review 15.08.2022 - Second Review

Table of contents

Introduction 4
Scope 4
Severity Definitions 5
Executive Summary 6
Checked Items 7
System Overview 10
Findings 11
Disclaimers 14

Introduction

Hacken OÜ (Consultant) was contracted by Astrobot (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/token-metrics/astrobot-smart-contracts
Commit:
610732a0072fdff907cf2538692d098e66d59f23
Technical Documentation:

Technical description

Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
-
Contracts:

```
File: ./contracts/nft.sol
SHA3: 18543ebb303ca9d9609b52e829980f87d8c0fe1dd9c0bbcd969e63ab53b0d2a6

File: ./contracts/sale.sol
SHA3: 65c1cde9a41a826fdbc6dfdebc2f0e76c50d6d1796fa0e1a9110469cdf53327c
```

Second review scope

Repository:
https://github.com/token-metrics/astrobot-smart-contracts
Commit:
08d763b2f8ae19e181f9cd37998fbe1f791b0162
Technical Documentation:

Technical description

Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
-
Contracts:

```
File: ./contracts/nft.sol
SHA3: 6a8d991ae47689d0b59ddaf8034dde27bb840aeb124b11238bd128d4d86575d5

File: ./contracts/sale.sol
SHA3: 21531e8b9fc14aec744b1a004086e9d59d94e1c97875625651b65c63516a9795
```

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 total Documentation Quality score is **10** out of **10**. Both functional requirements and technical documentation are provided.

Code quality

The total CodeQuality score is **6** out of **10**. NFT and sale logics are covered with tests. **Test coverage is 89.73%.** Cases testing successfully pausing or unpausing in the sale contract and successfully unpausing in the NFT contract are missing. There are style guide violations in file namings; they start with lowercase letters.

Architecture quality

The architecture quality score is **6** out of **10**. The minting functionality is distributed between two contracts. The whole logic can be implemented in a single contract, or there should be a full interface for the NFT contract.

Security score

As a result of the audit, the code contains **no** 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: **9.2**.

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

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
2 August 2022	2	3	2	0
15 August 2022	0	0	0	0

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.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
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.	Passed
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 self-destructible while 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.	Passed
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 Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Not Relevant

through tx.origin		authorization.	
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 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	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 or be predictable.	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.	Passed
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.	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.	Passed
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
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
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

Astrobot is an ERC721-based NFT project. It allows owners to have lifetime access to the Token Metrics platform.

- *Astrobot* – the main NFT contract with ERC721A format. The contract follows UUPS proxy pattern. Airdrop functionality is implemented where owners need to upload addresses and users need to claim airdrop tokens. The sale contract will call this contract for minting tokens during the sale.
- *Sale* – will be used for different phases of mint. The first two phases of sale will be for selected whitelisted users, and the third phase will be for everyone.

Privileged roles

- Owner: All control regarding the NFT and sale contract belongs to the owner. An owner can
    - enable/disable airdrop claim
    - update sale time/price/supply
    - update whitelisted addresses for airdrop
    - update whitelisting sale
- Multisig: Normal user of the system
    - can buy NFT in the public sale
    - a whitelisted user can participate in whitelisted/waitlisted sale

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. **Highly permissive role access**

The contract owner can withdraw all the ETH from the contract. This withdrawal can be done anytime without informing the users, leading to sudden balance changes.

This can lead to sudden ETH depletion in the contract.

Path: ./contracts/sale.sol : withdraw

Recommendation: Remove this functionality or inform users in public documentation.

Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

2. **Requirements violation**

While buying NFTs from the Sale contract, a user can send more ETH than needed. Then the excess funds are not refunded to the user.

This can lead to leftover ETH being lost.

Paths: ./contracts/sale.sol : mintWhitelisted()

./contracts/sale.sol : mintWaitlist()

./contracts/sale.sol : mintPublicSale()

Recommendation: The leftover amount should be refunded to the user if there are any leftovers.

Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

■■ Medium

1. **Phase supplies can be greater than max supply**

While declaring supplies, there are no checks if each of the phase supplies or the sum is greater than the declared maximum supply.

This can lead to minting more NFTs than the declared maximum supply.

Path: ./contracts/sale.sol : updateSaleSupply()

Recommendation: Implement related checks.

Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

2. **Unprotected initializer**

   Astrobot is an upgradeable contract that does not protect its *initialize* function. Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(address).

   This can lead to a change or destruction of the NFT contract.

   Path: ./contracts/nft.sol : initialize()

   Recommendation: Add a constructor to ensure *initialize* cannot be called on the logic contract.

   Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

3. **Unoptimized loop usage**

   The function updateAirdropAddresses uses a loop with user-supplied arrays. This will lead to Gas losses.

   This can lead to high Gas consumption.

   Path: ./contracts/nft.sol : updateAirdropAddresses()

   Recommendation: Cache arrays in a loop, save state variables to local memory, iterate the loop, and save changes to the state after the loop finish.

   Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

Low

1. **Missing zero address validation**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Paths: ./contracts/nft.sol : mint()

   ./contracts/sale.sol : initialize()

   Recommendation: Implement zero address checks.

   Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

2. **Default Variable Visibility**

   Variables' *userAirdropMints* and *userWhitelisted* visibilities are not specified. Specifying state variables' visibility helps to catch incorrect assumptions about who can access the variable.

   This makes the contract`s code quality and readability higher.

   Path: ./contracts/nft.sol : -

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Status: Fixed (Revised commit: 08d763b2f8ae19e181f9cd37998fbe1f791b0162)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.