



HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Agia International
Date: September 21th, 2022

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Agia International
Approved By	Evgeniy Bezuglyi SC Audits Department Head at Hacken OU
Type	ERC712 Token
Platform	EVM
Network	
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.unchainedpartners.com/
Timeline	29.08.2022 - 21.09.2022
Changelog	06.09.2022 - Initial Review 21.09.2022 - Second Review



Table of contents

Introduction	4
Scope	4
Severity Definitions	5
Executive Summary	6
Checked Items	7
System Overview	10
Findings	11
Disclaimers	12

Introduction

Hacken OÜ (Consultant) was contracted by Agia International (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/looksnap/AGIA>

Commit:

34b242c4e2768e8763298f6424a3aa33d4f0dbdc

Documentation:

Functional Requirements (attached for audit)

Integration and Unit Tests: No

Contracts:

File: ./FirstStage721_v3.sol

SHA3: 1fd82c39db9f440fec368f1a01e1fb4fa2162fab7e5c659005a817f54fc79120

File: ./SecondStage721_v3.sol

SHA3: 0957ff78b614686e6089f0105164d96627a17c121b1682616f7e1120a9e7f888

Second review scope

Repository:

<https://github.com/looksnap/AGIA>

Commit:

67134107e3160b8be1e3de3a9e4f5e033a3e818b

Documentation:

Functional Requirements (attached for audit)

Integration and Unit Tests: Yes

Contracts:

File: ./contracts/FirstStage721.sol

SHA3: ca0d3074b8f348143d975ce2e6a05ed084e7333a121fcec756f432d438f41863

File: ./contracts/SecondStage721.sol

SHA3: 3809cffc54d2c326e13021699df218180aba1487973fd6716ea8c08a1be5b8d0

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 [scoring methodology](#).

Documentation quality

The total Documentation Quality score is **3** out of **10**. Functional requirements are partially missed. A technical description is not provided.

Code quality

The total Code Quality score is **9** out of **10**.

- The repository lacks a description of how to install, test and deploy the code.

Test coverage

Test coverage of the project is **90%**.

- Not all *require* statements are tested.

Security score

As a result of the audit, the code contains **0** 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: **9.0**.

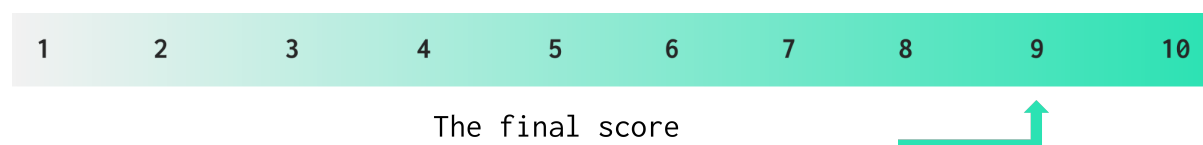


Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
6 September 2022	2	0	0	1
20 September 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.	Passed
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.	Passed
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.	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.	Not Relevant
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-Lev e1-2 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.	Failed
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.	Failed
Stable Imports	Custom	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

AGIA NFT is a system with the following contracts:

- *FirstStage* – ERC-721 token that can be bought only by EOA for a certain price. It can be airdropped.

It has the following attributes:

- Name: TestCollection
- Symbol: TC
- Maximal supply: 61 tokens.
- Minimal price per token: 1 ETH

- *SecondStage* – ERC-721 token that can be bought only by EOA for a certain price. It has multiple phases (whitelisted for buyers of *FirstStage* and public). It can be airdropped.

It has the following attributes:

- Name: TestCollectionPhase2
- Symbol: TCP2
- Maximal supply: 61 tokens.
- Minimal price per token: 1 ETH

Privileged roles

- For *FirstStage* token Owner can: toggle a whitelist [for the Whitelist NFT] on/off, and add/remove both addresses & NFT contracts from whitelist Pause/unpause transfers (including any buys/trades); Set prices for individual NFTs; Withdraw all Ether from contract; Withdraw any ERC20 tokens from contract; Update the baseURI (e.g., to do an optional reveal); Airdrop whitelist NFTs to a target address; Sets a price to 0.
- For *SecondStage* token Owner can: add/remove time from whitelist for individual NFTs; Pause/unpause transfers (including any buys/trades); Set prices for individual NFTs; Withdraw all Ether from contract; Withdraw any ERC20 tokens from contract; Airdrop an NFT to an address (for free); Update the baseURI (e.g., to do an optional reveal); Transfer arbitrary NFTs owned by users (trust in the doxxed Owner is key here); Change the minimum price.

Findings

■■■■ Critical

1. Requirements violation

The requirements state that Whitelist NFTs are never burned. However, the FirstStage token is ERC721Burnable. This allows the tokens to be burned and may produce additional scarcity.

The code should not violate the requirements provided by the Customer.

Path: ./FirstStage721_v3.sol

Recommendation: Stop inheriting from ERC721Burnable contract.

Status: Fixed (6713410)

■■■ High

No high severity issues were found.

■■ Medium

No medium severity issues were found.

■ Low

1. Floating pragma

Contracts files use floating pragma ^0.8.14

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths: ./FirstStage721_v3.sol, ./SecondStage721_v3.sol

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

Status: Fixed (6713410)

2. Gas optimization

Using best Solidity code practices saves some Gas and reduces the transaction cost.

Paths: ./FirstStage721_v3.sol:addWhitelistedUsers(),
./FirstStage721_v3.sol:removeWhitelistedUsers()

Recommendation: Cache array length.

Status: Fixed (6713410)

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.