HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: MAJR INC
Date: JUL 26th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for MAJR INC
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	ERC721A token; NFT
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://majr.io
Timeline	28.06.2022 - 26.07.2022
Changelog	28.06.2022 - Initial Review 26.07.2022 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction 4

Scope 4

Severity Definitions 5

Executive Summary 6

Checked Items 7

System Overview 10

Findings 11

Disclaimers 13

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by MAJR INC (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/MAJR-Inc/majr-smart-contracts
Commit:
15d0c90ec154cfab6cb2e30c394cc41f25e5db06
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://drive.google.com/file/d/17CBalDs9PajVA16r1D7GDxXZgOAzuo2O/view

Type: Technical description
https://drive.google.com/file/d/17CBalDs9PajVA16r1D7GDxXZgOAzuo2O/view?usp=sharing

Type: Functional requirements
https://drive.google.com/file/d/17CBalDs9PajVA16r1D7GDxXZgOAzuo2O/view?usp=sharing

Integration and Unit Tests: Yes
https://github.com/MAJR-Inc/majr-smart-contracts/tree/main/test

Contracts:
File: ./contracts/MajrNFT.sol
SHA3: 9d5b551d69d8a5a859544e74af4905f1099471dcaa950a288ab7f6f1e3ba2d64

File: ./contracts/Splitter.sol
SHA3: 8630286a8f1c09a3453db7a21935210660ddf5e70983d084f4ca2beae417652b

Second review scope

Repository:
https://github.com/MAJR-Inc/majr-smart-contracts
Commit:
c9a22e4b222326b3925efd28a150097951f878ff
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://drive.google.com/file/d/17CBalDs9PajVA16r1D7GDxXZgOAzuo2O/view

Type: Technical description
https://drive.google.com/file/d/17CBalDs9PajVA16r1D7GDxXZgOAzuo2O/view?usp=sharing

Type: Functional requirements
https://drive.google.com/file/d/17CBalDs9PajVA16r1D7GDxXZgOAzuo2O/view?usp=sharing

Integration and Unit Tests: Yes
https://github.com/MAJR-Inc/majr-smart-contracts/tree/main/test

Contracts:
File: ./contracts/MajrNFT.sol
SHA3: 252470c2f5521335609ca40cf44b47537694276de9fe3f4012abf3d0c39a9129

File: ./contracts/Splitter.sol
SHA3: 7351a96b1c7065fc19247983aea892bb64a6b1c2472dcf71180e9bc9c44bd016

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

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

Documentation quality

The total Documentation Quality score is **7** out of **10**. Functional requirements are partially missed. A technical description is not provided.

Code quality

The total CodeQuality score is **8** out of **10**. Several functions with incorrect access levels were found.

Architecture quality

The architecture quality score is **6** out of **10**. A push-based system is used when NFTs are sold, causing unnecessary Gas costs in the “mint” and “mintWithReferer” functions.

Security score

As a result of the audit, the code contains **1** low severity issue. 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: **9.1**.

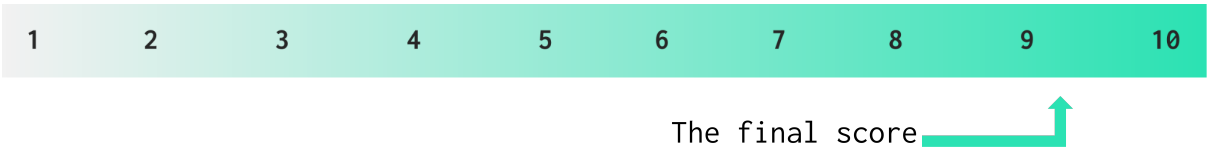

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
28 Jun 2022	7	1	1	0
22 Jul 2022	1	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.	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.	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 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.	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	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 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.	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.	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	**Custom**	Transaction execution costs should not	Passed

Loops		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.	
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

Majr is a mixed-purpose system with the following contracts:

- *MajrNFT* — ERC-721 Gas optimized NFT smart contract that sets max supply value and tokenBaseURI and contractURI for interaction with OpenSea.
  It has the following attributes:
  - Name: MAJR ID NFT
  - Symbol: MAJR
- Splitter - is a simple, ownable smart contract where only the owner can set up referral and split address with the corresponding amount value that represents the part of msg.value will be sent to that addresses as receivers. When NFT are minted, the ETH payment is split 3 main ways and in the case of a referral, split a 4th way referral addresses proportionally according to the amount settings.

When users purchase the NFT from a referred person (the referral), the ETH is split into 4 main wallet addresses (Earnings, DAO, MAJR Inc Revenue, the Referral wallet). The referral tech works with an ETH payment splitter and wallet attribution using a JavaScript link shortener.

Privileged roles

- The owner of the *MajrNFT* contract can transfer ownership to another address. The owner can leave the contract without an owner and remove functionality that is only available to the owner.
- The owner of the *MajrNFT* contract can pause/unpause smart contract's functionality, set metadata: 'baseUri' and 'ContractUri' for interaction with OpenSea. The owner can set referral and split addresses that are a core part of MajrNFT contract's functionality.

Risks

- In case of an admin keys leak, an attacker can transfer ownership to another address or remove the owner of the contract at all. Attackers can set up their 'referral' and 'split' addresses which can lead to loss of assets.

Findings

Critical

No critical severity issues were found.

High

1. Denial of service.

The *transfer* function is used to transfer native currency.

If the target address is a contract with fallback functionality, the execution will fail due to the Gas limit of the *transfer* function.

File: ./contracts/Splitter.sol

Contract: Splitter.sol

Functions: split, referralSplit

Recommendation: Change push transfer pattern to pull transfer pattern.

Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

Medium

1. Redundant Gas usage when calling external functions.

The external function should be declared as public for internal calling to reduce Gas fees.

Functions ‘mint’ and ‘mintWithReferral’ call external functions ‘split’ and ‘referralSplit’.

File: ./contracts/MajrNFT.sol

Contract: MajrNFT.sol

Functions: mint, mintWithReferrer

Recommendation: Declare ‘mint’ and ‘mintWithReferral’ functions ‘public’ instead of ‘external’.

Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

Low

1. Boolean equality.

Boolean constants can be used directly and do not need to be compared to true or false. Boolean equality can lead to errors in the logic of code.

Detects the comparison to boolean constants ‘_checkForReferralAddress’ in ‘Splitter’ contract.

File: ./contracts/Splitter.sol

Contract: Splitter.sol

Function: _checkForReferralAddress

Recommendation: Remove the equality to the boolean constant.

Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

2. **Different pragma directives are used.**

   Usage of different pragma directives can lead to errors in logic of code.

   Splitter.sol uses ^0.8.11 Solidity version but Ownable.sol and Context.sol use ^0.8.0 Solidity version.

   File: ./contracts/Splitter.sol

   Contract: Splitter.sol

   Recommendation: Use a specific Solidity version for both contracts.

   Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

3. **Unused state variable.**

   The code should not contain unused variables. If a local variable is declared but not used, it is dead code and should be removed.
   Doing so will improve maintainability because developers will not wonder what the variable is used for.

   State variables `beneficiary` and ‘incentive’ are never used.

   File: ./contracts/Splitter.sol

   Contract: Splitter.sol

   State variables: beneficiary, incentive

   Recommendation: Remove unused state variables.

   Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

4. **Conformance to Solidity naming conventions.**

   Solidity defines a naming convention that should be followed. Some state variables are not the mixed case.

   ‘Splitter’ contract contains several functions that are declared using the wrong naming convention.

   File: ./contracts/Splitter.sol

   Contract: Splitter.sol

   State variables: _splitAddresses _splitAmounts _referralAddresses _referralAmounts

Recommendation: Follow the Solidity naming convention.

Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

5. **State variables are declared as not constant.**

   Constant state variables should be declared constant to save Gas. Using ‘constant’ keywords in the variable declaration will make some optimization for Gas usage.

   Splitter contract contains few state variables such as beneficiary, incentive that can be declared as ‘constant’.

   Declaration state variables without 'constant' keywords can lead to more Gas usage.

   File: ./contracts/Splitter.sol

   Contract: Splitter.sol

   State variables: beneficiary, incentive

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

   Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

6. **Public function that could be declared external.**

   ‘Public’ functions that are never called by the contract should be declared as ‘external’ to save Gas. Using a public modificator instead of external leads to more Gas usage during the execution of a transaction.

   There are 2 functions of ’MajrNFT’’ contract that could be declared as ’external’ instead of ‘public’ : mint, mintWithReferrer.

   File: ./contracts/MajrNFT.sol

   Contract: MajrNFT.sol

   Functions: mint, mintWithReferrer.

   Recommendation: Use the ‘external’ attribute for functions never called from the contract.

   Status: Reported

7. **Local variable shadowing.**

   State variables should not be shadowed.

   The constructor of MajrNFT shadows such variables as ‘name’ and ‘symbol’ in Erc721’s contract.

Shadowing variables can lead to errors in the logic of code.

File: ./contracts/MajrNFT.sol

Contract: MajrNFT.sol

Function: constructor of MajrNFT contract.

Recommendation: Rename the local variables that shadow another component.

Status: Fixed (Revised commit: c9a22e4b222326b3925efd28a150097951f878ff)

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.