HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Emergent Games
Date: Jul 22nd, 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 Emergent Games
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC721 standard
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://yop.finance
Timeline	14.07.2022 - 22.07.2022
Changelog	22.07.2022 - Initial Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Emergent Games (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/chiru-labs/ERC721A-Upgradeable
Commit:
677d1976d75d25fbf994c2f6b2ae3cb025cd18a0
Technical Documentation:
Type: Technical description
ERC721A Docs

Type: Functional requirements
EIP-20 Token Standard

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/ERC721A__Initializable.sol
SHA3: da731d3817fd9e98583ead2943d1725eda150afe22306835971b478af2642dc9

File: ./contracts/ERC721A__InitializableStorage.sol
SHA3: 702cb7ae7c1491fbef0e97a30145b9fa673b747b452537c0cacbf08e7d53df72

File: ./contracts/ERC721AStorage.sol
SHA3: 95704cf15a671458c8414587cc2ccb22bbb914a9880cea68c813d40a115ecc0a

File: ./contracts/ERC721AUpgradeable.sol
SHA3: 3b55cf8181ba5789e41e6d9fe199108f39feb668349f774d566e8dc94f455e8f

File: ./contracts/extensions/ERC721ABurnableUpgradeable.sol
SHA3: aef522d20f4737eef0a6b40319078d44d988f94009469f0ae6380aba343ed51c

File: ./contracts/extensions/ERC721AQueryableUpgradeable.sol
SHA3: 1272ed22019c87d3b093b9c526f253dc30bf097e4db5fff0ce6ff12048d3485e

File: ./contracts/extensions/IERC721ABurnableUpgradeable.sol
SHA3: b0b9815dea448719cdbcf81ce6f1ec3778084aac9b3377771e051c94c2e5fd85

File: ./contracts/extensions/IERC721AQueryableUpgradeable.sol
SHA3: 8e2a7123e9d5b48334dcd04604dc786dc05720cefee568b175266848dcd1ce74

File: ./contracts/IERC721AUpgradeable.sol
SHA3: ced6740abbb65335647502e74e1c7a9e0fac5f1da6bb606d55caea01cf21459d

File: ./contracts/interfaces/IERC721ABurnableUpgradeable.sol
SHA3: 24774e980dc334bd12018d0252e4ef9add4cf9e3c4bba18c540ad9a647f8febb

File: ./contracts/interfaces/IERC721AQueryableUpgradeable.sol

```
SHA3: 24a252a77468b911a57ec5cf8500d0a9e748043094c75d4da3baa7a6210a8059

File: ./contracts/interfaces/IERC721AUpgradeable.sol
SHA3: 0eb958d12fc465fc6607a07a5d6e803fa15ee992488f1f718c37f9d2ec49ea51
```

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 **9** out of **10**. The code is well documented. Full technical requirements are implicitly provided through the EIP-20 Token Standard proposal. However, the function "_checkContractOnERC721Recieved" contains complex code, which is recommended to document better.

Code quality

The total CodeQuality score is **8** out of **10**. The code follows official language style guides and is mostly covered with unit tests.

Architecture quality

The architecture quality score is **10** out of **10**. Clean and clear architecture and well-configured development environment.

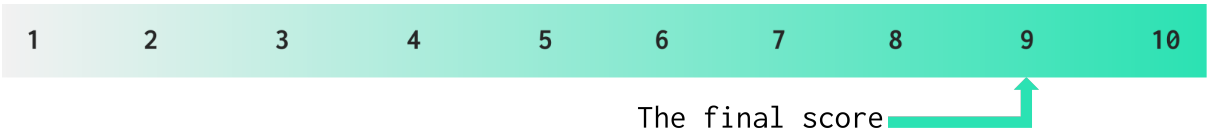
Security score

As a result of the audit, the code contains **1** medium and **2** low severity issues. The security score is **9** 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**.



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.	Failed
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.	Not Relevant
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	Passed

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.	Not Relevant
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.	Not Relevant
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.	Not Relevant
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of	Not Relevant

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

System Overview

ERC721A enables a near constant Gas cost for batch minting via a lazy-initialization mechanism.

Token IDs are minted in sequential order (e.g. 0, 1, 2, 3, …).

Regardless of the quantity minted, the _mint function only performs 3 SSTORE operations:

- Initialize the ownership slot at the starting token ID with the address.
- Update the address' balance.
- Update the next token ID.

The repository being audited is automatically transpiled from the main ERC721A non-upgradeable repo.

The files in the scope:

- **ERC721AUpgradeable.sol** - implementation of ERC721 Non-Fungible Token Standard, including the Metadata extension. Built to optimize for lower Gas during batch mints.
- **ERC721AStorage.sol** - a diamond storage contract to be used with the ERC721Upgradeable above.
- **ERC721A__Initializable.sol** - a base contract to aid in writing upgradeable diamond facet contracts, or any kind of contract that will be deployed behind a proxy. Since proxied contracts do not use a constructor, it is common to move constructor logic to an external initializer function, usually called `initialize`. It then becomes necessary to protect this initializer function so it can only be called once. The {initializer} modifier provided by this contract will have this effect.
- **ERC721A__InitializableStorage.sol** - a base storage for the initialization function for upgradeable diamond facet contracts.
- **ERC721AQueryableUpgradeable.sol** - ERC721A subclass with convenience query functions.
- **ERC721ABurnableUpgradeable.sol** - ERC721A Token that can be irreversibly burned (destroyed).

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

Empty initializer

The initializers for the “ERC721ABurnableUpgradeable” and “ERC721AQuaryableUpgradeable” contracts are empty functions.

Functions: __ERC721ABurnable_init, __ERC721ABurnable_init_unchained, __ERC721AQueryable_init, __ERC721AQueryable_init_unchained

Recommendation: Implement the initialization logic or remove redundant code.

Status: New

Low

1. **Functions should be declared external**

   In order to save Gas, public functions that are never called in the contract should be declared as external.

   Paths: ERC721AUpgradeable.approve, ERC721AUpgradeable.getApproved, ERC721AUpgradeable.name, ERC721AUpgradeable.safeTransferFrom, ERC721AUpgradeable.setApprovalForAll, ERC721AUpgradeable.supportsInterface, ERC721AUpgradeable.symbol, ERC721AUpgradeable.tokenURI, ERC721AUpgradeable.totalSupply, ERC721ABurnableUpgradeable.burn

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

   Status: New

1. **Pragma allows old versions**

   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.

   Contracts: WithInit.sol, ERC721Storage, ERC721A__Initializable, ERC721A__InitializableStorage, ERC721ReceiverMockStorage, StartTokenIdHelperStorage

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

   Status: New

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.