HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Acta Finance/P2P Solutions LTD
Date: Apr 15th, 2021

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 Acta Finance/P2P Solutions LTD.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC721 token
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://actafi.org/
Timeline	12.04.2022 - 04.05.2021
Changelog	15.04.2022 - Initial Review 29.04.2022 - Second Review 04.05.2022 - Third 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 Acta Finance/P2P Solutions LTD (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/P2P-Finance/acta-nft-token.git
Commit:
428d82167ead9932b7b30068b279c2369a3cad9b
Technical Documentation: Yes
JS tests: No
Contracts:
./contracts/ActaNftToken.sol

Second review scope

Repository:
https://github.com/P2P-Finance/acta-nft-token.git
Commit:
bc543fa99cf552cee07093ea8ffa58bda4b1a1a3
Technical Documentation: Yes
JS tests: Yes
Contracts:
./contracts/ActaNftToken.sol

Third review scope

Repository:
https://github.com/P2P-Finance/acta-nft-token.git
Commit:
af10608bcaf87ce940ffe7f6c3dbe512c11857df
Technical Documentation: Yes
JS tests: Yes
Contracts:
./contracts/ActaNftToken.sol

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

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 measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided a whitepaper with functional requirements. The whitepaper does not contain any information about NFTs. The total Documentation Quality score is **5** out of **10**.

Code quality

The total CodeQuality score is **5** out of **10**. No unit tests were provided.

As a result of the third review, CodeQuality score was changed to **10** out of **10**. Good unit tests coverage were provided.

Architecture quality

The architecture quality score is **10** out of **10**. The project has clear and clean architecture.

Security score

As a result of the audit, security engineers found **2** high, and **0** medium severity issues. The security score is **0** out of **10**.

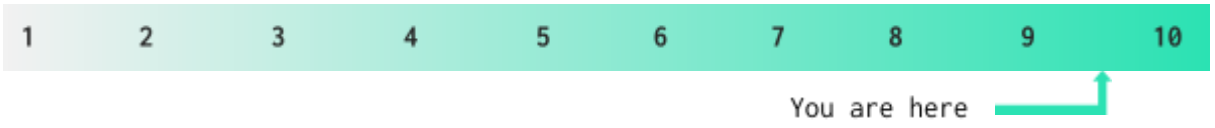
As a result of the second review, security engineers found 2 new low severity issues. **2** high severity issues from the previous revision were fixed. As a result, the code contains no issues. The security score is **10** out of **10**.

As a result of the third review, security engineers found no new issues. **2** low severity issues from the previous revision were fixed. As a result, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Issues overview” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.5**



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

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.	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.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until 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
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
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.	Passed
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.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	Passed
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 Loops	**Custom**	Transaction execution costs should not 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.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with requirements provided by the Customer,	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed

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

System Overview

Acta Finance is a DeFi ecosystem that brings innovation to the DeFi industry by motivating users to build a referral network that rewards user network, and user activity, through the referral system and address milestones. ActaFi Swap is a cross-chain liquidity aggregator that offers margin trading that receives the margin from peer-to-peer lending offers, introducing a new investment opportunity for the users.

- ActaNftToken — a contract that implements the ERC721 contract. Minting is available only for allowed by contract owner roles with any URI for NFT image.

Privileged roles

- Owner can assign roles for NFT minting
- Owner can pause the contract in case of any errors to stop minting

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

Findings

Critical

No critical severity issues were found.

High

1. **NFT id's collision is possible**

   SafeMint function increments the NFT id counter only after the mint. As a result, safeMint will be executed with an already existing NFT id, which will lead to an exception. The functionality of the function will be blocked.

   Contracts: ActaNftToken.sol

   Function: safeMint

   Recommendation: Call *_tokenIdCounter.increment()* before *uint256 tokenId = _tokenIdCounter.current();*

   Status: Fixed (bc543fa99cf552cee07093ea8ffa58bda4b1a1a3)

2. **Mint is available for everyone**

   Mint function is available without any restrictions, so anyone can mint a token.

   Contracts: ActaNftToken.sol

   Function: mintToken

   Recommendation: Limit access to function to prevent unauthorized access.

   Status: Fixed (bc543fa99cf552cee07093ea8ffa58bda4b1a1a3)

Medium

No medium severity issues were found.

Low

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

   Public functions that are never called by the contract should be declared external to save Gas.

   Contracts: ActaNftToken.sol

   Functions: pause, unpause, mintToken, safeMint

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

   Status: Fixed (bc543fa99cf552cee07093ea8ffa58bda4b1a1a3)

2. **Improper Input Validation.**

As the mint function has no restricted access - it could be executed by anyone, so nft URI could be any string. For operation with such data from blockchain input validation technique should be applied for checking potentially dangerous inputs to ensure that the inputs are safe for processing within the code or when communicating with other components. When software does not validate input properly, an attacker can craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Contracts: ActaNftToken.sol

Function: mintToken

Recommendation: Confirm that public access to the function is expected behavior, and input validation technique would be applied in case of operation with this data on back-end.

Status: Reported

3. **Redundant import**

The contains import, that is not used and could be deleted.

Contracts: ActaNftToken.sol

Recommendation: Remove `IERC20.sol` import.

Status: Fixed (af10608bcaf87ce940ffe7f6c3dbe512c11857df)

4. **Floating Pragma.**

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. 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: ActaNftToken.sol

Recommendation: Use a fixed version of the compiler (^ symbol should be removed from Pragma)

Status: Fixed (af10608bcaf87ce940ffe7f6c3dbe512c11857df)

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

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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