SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

FREEE

AUDIT

SECURITY ASSESSMENT

15. October, 2024

FOR

Introduction

SolidProof.io is a brand of the officially registered company FutureVisions Deutschland, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement"or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	Freee
Website	https://www.freee.xyz/
About the project	Freee.xyz is a multi-chain NFT platform where you can create, share, and collect to earn.
Chain	Bitlayer
Language	Solidity
Codebase	https://www.btrscan.com/address/0x16c6ffe6782769ca6abceb7258a08c13f9b54c36?tab=Contract
Commit	N/A
Unit Tests	Not Provided

Social Medias

Telegram	https://t.me/freeexyz
Twitter	https://x.com/freeexyz
Facebook	N/A
Instagram	N/A
GitHub	N/A
Reddit	N/A
Medium	N/A
Discord	https://discord.gg/freee
YouTube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Change Log
v1.0	15. October 2024	· Layout Project · Automated/ Manual-Security Testing · Summary

Note – The following audit report presents a comprehensive security analysis of the smart contract utilized in the project that includes outside manipulation of the contract's functions in a malicious way. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as we did not functionally test it. This includes internal calculations in the formulae used in the contract.

File Overview

The Team provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/ERC721Collection.sol	Ad9e182ace1f52ed8920671d45da4fe48aedb221
contracts/ERC721CollectionProxy.sol	b2e0ecae7708de4256a7c2253a642fa5df127a7a

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages.

Used code from other Frameworks/Smart Contracts.

Dependency / Import Path	Count
erc721a-upgradeable/contracts/ERC721AUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/interfaces/IERC721Upgradeable.sol	1
erc721a-upgradeable/contracts/IERC721AUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/interfaces/IERC2981Upgradeable.sol	1
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/token/ERC1155/extensions/ERC1155BurnableUpgradeable.sol	1
@openzeppelin/contractsupgradeable/utils/cryptography/MerkleProofUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	1
@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol	1

Dependency / Import Path	Count
./interfaces/IMetadataRenderer.sol	1
./interfaces/IERC721Drop.sol	1
./interfaces/IOwnable.sol	1
./interfaces/IERC4906.sol	1
./interfaces/IFactoryUpgradeGate.sol	1
./utils/OwnableSkeleton.sol	1
./utils/FundsReceiver.sol	1
./utils/Version.sol	1
./utils/PublicMulticall.sol	1
./storage/ERC721DropStorageV1.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract'sstate. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be needed within visibility modifier, such as public, private or internal, which determines the access level of the variable.

Components

📝 Contracts	📚 Libraries	🔍 Interfaces	🎨 Abstract
1	0	0	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐 Public	💰 Payable
32	4

External	Internal	Private	Pure	View
20	36	0	1	14

StateVariables

Total	🌐 Public
9	3

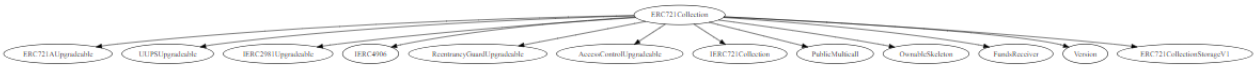
Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
^0.8.10	----------	Yes		----------

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
yes					

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methodsfrom another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit the vulnerability and the impact of that event on the organization or system.The risk level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executingthe contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers andsmart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   a. Reviewing the specifications, sources, and instructions provided to
   SolidProof to ensure we understand the size, scope, and functionality of the
   smart contract.
   b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
   c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
   a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
   b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Upgradeability

Contract is an upgradable	✗ Deployer can update the contract with new functionalities.
Description	The contract is an upgradeable contract. The Deployer is able to change or add any functionalities to the contract after deploying.
Comment	The owner can update the contract with new functionalities into the contract.

File/Line(s): L171-218

Codebase: ERC721Collection.sol

```
    /// @param _metadataRendererInit Renderer data initial contract
    function initialize(
        string memory _contractName,
        string memory _contractSymbol,
        address _initialOwner,
        address payable _fundsRecipient,
        uint64 _collectionSize,
        uint16 _royaltyBPS,
        bytes[] calldata _setupCalls,
        bool _tradingLocked,
        bool _revealed,
        IMetadataRenderer _metadataRenderer,
        bytes memory _metadataRendererInit,
        bytes memory _dynamicMetadataInit
    ) public initializer {
        // Setup ERC721A
        __ERC721A_init(_contractName, _contractSymbol);
        // Setup access control
        __AccessControl_init();
        // Setup re-entracy guard
        __ReentrancyGuard_init();
        // Setup the owner role
        _setupRole(DEFAULT_ADMIN_ROLE, _initialOwner);
        // Set ownership to original sender of contract call
        _setOwner(_initialOwner);

        if (_setupCalls.length > 0) {
            // Setup temporary role
            _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
            // Execute setupCalls
            multicall(_setupCalls);
            // Remove temporary role
            _revokeRole(DEFAULT_ADMIN_ROLE, msg.sender);
        }

        // Setup config variables
        config.collectionSize = _collectionSize;
        config.metadataRenderer = _metadataRenderer;
        config.royaltyBPS = _royaltyBPS;
        config.fundsRecipient = _fundsRecipient;
        config.revealed = _revealed;
        config.lockBeforeMintOut = _tradingLocked;

        if (config.royaltyBPS > MAX_ROYALTY_BPS) {
            revert Setup_RoyaltyPercentageTooHigh(MAX_ROYALTY_BPS);
        }

        _metadataRenderer.initializeWithData(_metadataRendererInit, _dynamicMetadataInit);
    }
```

Ownership

Contract ownership is not renounced.	❌ **The ownership is not renounced.**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations.
Comment	N/A

Note – *The contract cannot be considered as renounced till it is not deployed or having some functionality that can change the state of the contract.*

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner can not mint new tokens.	✅ **The owner cannot mint new tokens.**
Description	The owner is able to mint new tokens once the contract is deployed.
Comment	N/A

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens.**
Description	The owner is not able to burn tokens without any allowances.
Comment	Anyone can burn tokens.

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses.	✅ **The owner cannot blacklist wallets.**
Description	The owner cannot blacklist wallets from transferring of tokens.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort indeveloping and maintaining the contract.

Contract owner cannot set fees more than 25%.	✅ **The owner cannot set fees more than 25%.**

Description	The owner cannot set fees of more than 25%.
Comment	N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. When token or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Contract owner cannot lock function.	✅ **The owner cannot lock function.**
Description	The owner cannot lock the contract.
Comment	N/A

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example,if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
ERC721Collection.sol	➢ The public sale active can purchase ➢ The public sale active can purchase with comment. ➢ The presale active purchase presale. ➢ The admin/only role can mint a specified quantity of NFTs. ➢ The owner can set the new owner's address. ➢ The owner can set/call metadata renderer. ➢ The owner can set sale configurations. ➢ The owner can reveal the collection. ➢ The owner can set a trading lock. ➢ The owner can set the royalty. ➢ The owner can set the fund recipient address.

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement

- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Result

Critical Issues

No critical issues

High Issues

No high issues

Medium Issue

No medium issues

Low Issue

#1 | Floating pragma solidity version.

File	Severity	Location	Status
ERC721Collection	Low	L2	Open

Description – Adding the constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

#2 | Remove SignedMath and Math library.

File	Severity	Location	Status
MathUpgradeable	Low	L9-339	Open
SignedMathUpgradeable	Low	L9-43	Open

Description – The compiler version above 0.8.0 has the ability to control arithmetic overflow/underflow. It is recommended to remove the unwanted code in order to avoid high gas fees.

Informational Issue

No informational issues

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY