SOLIDProof

*Bring trust into your projects*

**Blockchain Security | Smart Contract Audits | KYC Development | Marketing**

MADE IN GERMANY

# BSCS NFT

# AUDIT

## SECURITY ASSESSMENT

**13. December, 2023**

**FOR**

BSCS

# Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, 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 | BSCS |
|---|---|
| Website | https://bscs.finance/ |
| About the project | The fully decentralized protocol for launching new ideas. An all-in-one Incubation Hub with a full-stack Defi platform across all main blockchain networks. We provide exclusive services including IDO/INO Launchpad, Yield farming, NFT Auction, Marketplace, and DEXswap |
| Chain | BSC |
| Language | Solidity |
| Codebase Link | Provided as Files (Private Repo) |
| Commit | N/A |
| Unit Tests | Not Provided |

## Social Medias

| | |
|---|---|
| Telegram | https://t.me/bscstation |
| Twitter | https://twitter.com/bscstation |
| Facebook | N/A |
| Instagram | N/A |
| Github | N/A |
| Reddit | N/A |
| Medium | N/A |
| Discord | N/A |
| Youtube | N/A |
| TikTok | N/A |
| LinkedIn | N/A |

# Audit Summary

| Version | Delivery Date | Changelog |
|---|---|---|
| v1.0 | 12. December 2023 | • Layout Project<br>• Automated- /Manual-Security Testing<br>• Summary |
| v1.1 | 13. December 2023 | • Reaudit |

**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/MintNFT.sol | 08ffe8fed0dfd3926fcf852b1f1319ec40a05a19 |
| contracts/NFTMarketplace.sol | 5114f6c21115ab33ff149c2fceaf5ac13009d5f3 |
| contracts/NFTEnglishAuction.sol | f24d582ca8f4122da3fd67048cfa721b0ca480c9 |

*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 (direct imports).*

| Dependency / Import Path | Count |
|---|---|
| @openzeppelin/contracts/access/Ownable.sol | 3 |
| @openzeppelin/contracts/security/Pausable.sol | 1 |
| @openzeppelin/contracts/security/ReentrancyGuard.sol | 1 |
| @openzeppelin/contracts/token/ERC20/IERC20.sol | 3 |
| @openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol | 2 |
| @openzeppelin/contracts/token/ERC721/ERC721.sol | 1 |
| @openzeppelin/contracts/token/ERC721/IERC721.sol | 2 |
| @openzeppelin/contracts/token/ERC721/IERC721Receiver.sol | 2 |
| @openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol | 1 |
| @openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol | 1 |
| @openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol | 1 |
| @openzeppelin/contracts/utils/Strings.sol | 1 |
| @openzeppelin/contracts/utils/math/SafeMath.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

# Audit Information

## Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit 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 executing the 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 and smart 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 not an upgradeable | ✅ Deployer cannot update the contract with new functionalities |
|---|---|
| Description | The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying. |
| Comment | N/A |

## Ownership

| **The ownership is not renounced** | ❌ **The owner is not renounce** |
|---|---|
| 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:<br>• Centralizations<br>• The owner has significant control over contract's operations |
| Comment | N/A |

**Note** - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

# 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 cannot mint new tokens** | ✅ **The owner cannot mint new tokens** |
|---|---|
| Description | The owner is not 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 burn tokens without any allowances. |
| Comment | N/A |

## 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 addresses** |
|---|---|
| Description | The owner is not able blacklist addresses to lock funds. |
| 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 in developing and maintaining the contract.*

| **Contract owner can set fees greater than 25%** | ❌ **The owner able to set unfair fee** |
|---|---|
| Description | For example, a decentralized exchange (DEX) or a NFT MarketPlace smart contract may charge a fee for each trade executed on the platform. This fee can be set by the owner of the contract and may be a percentage of the trade value or a flat fee.<br>In other cases, the owner of the smart contract may set fees for accessing or using certain features of the contract. For instance, a subscription-based service smart contract may charge a monthly or yearly fee for access to premium features.<br>Overall, fees set by the owner of a smart contract can provide an additional source of revenue for the contract's owner and can help to ensure the sustainability of the contract over time. |
| Example | Our assumption is that the owner can adjust the creator, and service fees up to 100%. If the fee is set to 100%, it implies that the full amount of tokens you intend to send will be sent to the address specified as the fees recipient in the contract. This implies that the recipient will never have the intended amount of tokens in their wallet as it has all been used up in paying for the transfer fee. |
| 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 tokens 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 can lock the user funds** | ❌ **The owner is able to lock the contract** |
|---|---|
| Description | Locking the contract means that the owner is able to lock any funds of addresses that they are not able to transfer bought tokens anymore. |
| Example | An example of locking is by pausing the minting contract or setting the fees more than or 100%. |
| Comment | N/A |

## 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's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.*

# Components

| 📝 Contracts | 📚 Libraries | 🔍 Interfaces | 🎨 Abstract |
|---|---|---|---|
| 3 | 0 | 2 | 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 |
|---|---|
| 42 | 1 |

| External | Internal | Private | Pure | View |
|---|---|---|---|---|
| 28 | 46 | 1 | 2 | 11 |

# StateVariables

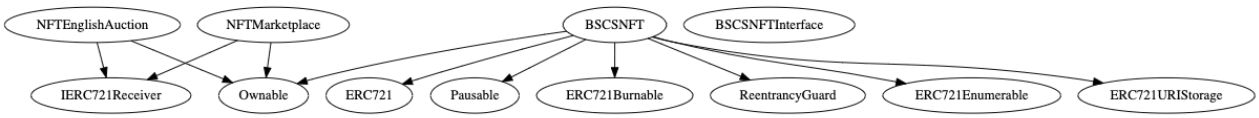
| Total | 🌐 Public |
|---|---|
| 23 | 20 |

## Capabilities

| Solidity Versions observed | Transfers ETH | 💰 Can Receive Funds | 🖥️ Uses Assembly | 💣 Has Destroyable Contracts |
|---|---|---|---|---|
| ^0.8.1<br>^0.8.2 | Yes | 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 methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.*



# 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 |
|---|---|
| **MintNFT** | • Set Mint Fees to any arbitrary value<br>• Set MintFee token address, that can be changed at any time<br>• Set Fee recipient addressand Base URI<br>• Pause/Unpause transfer function |
| **MintEnglishAuction** | • Set Treasury Address<br>• Set Service and Creator fees to any arbitrary value including 100%<br>• Whitelist NFT contracts and Currency Contracts<br>• Withdraw Stuck tokens from the contract but it is also possible for the owner to withdraw the funds deposited by the bidders |
| **NFTMarketPlace** | • Set Treasury Address<br>• Set Service and Creator fees to any arbitrary value including 100%<br>• Whitelist NFT contracts and Currency Contracts |

## 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 Results

## Critical issues

**No critical issues**

## High issues

**No high issues**

## Medium issues

### #1 | The Fees can be 100% or more

| File | Severity | Location | Status |
|---|---|---|---|
| NFTEnglishAuction | Medium | L93, 98 | ACK |
| NFTMarketplace | Medium | L192, 197 | ACK |

**Description -** The contract owner can set the service and creator fee without any restrictions, which means the fees could also be set to 100%. The repercussions of this will be grave as the complete transaction amount will be transferred to the treasury or creator address.

**Remediation -** We recommend putting an upper range on the fees where it cannot surpass a maximum threshold of 25%

### #2 | Owner can drain tokens

| File | Severity | Location | Status |
|---|---|---|---|
| NFTEnglishAuction | Medium | L500 | ACK |

**Description -** The owner of the contract can drain the complete balance of the contract by calling this function. Although the function is intended to withdraw only stuck tokens from the contract, there are no checks to impose it. However, it is also possible for the owner to withdraw the bidding amount deposited by the users in the contract while bidding for NFTs

**Remediation -** Make sure that it is not possible to take out the tokens that are deposited for bidding.

### #3 | Missng “isContract” check

| File | Severity | Location | Status |
|---|---|---|---|
| NFTEnglishAuction | Medium | L264 | Fixed |

**Description** - The contract has no checks to verify whether an EOA or contract calls the bid function. This could be very risky after the contract goes live, as it opens up the contract to “botting”, which means that a malicious user can create a contract that will call the bid function as soon as any other user bids.
Since there is no step-by-step increment system for the bidding amount, the attacker can call the bid function with just 1 extra token from the previous bid.

**Remediation -** We recommend putting a check to verify that the caller of the BID function must be an EOA. Moreover, we also recommend implementing a cooldown period in the bidding function so that the function cannot be called recursively without any restrictions.

## Low issues

### #1 | Missing Events

| File | Severity | Location | Status |
|---|---|---|---|
| NFTEnglishAuction | Low | L103, 113 | ACK |

**Description** - Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes.

### #2 | Missing Check

| File | Severity | Location | Status |
|---|---|---|---|
| NFTEnglishAuction | Low | L324, 384 | ACK |

**Description** - Ensure the bid owner cannot call the claim function more than once. However, when the auction status is being set at the end of the claim functions, it should also be checked in the function to ensure it is only callable once.

# Informational issues

## #1 | Floating Pragma

| File | Severity | Location | Status |
|---|---|---|---|
| All | Informational | N/A | ACK |

**Description** - The contracts should be deployed with the same compiler version and flag that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

## #2 | General Recommendation

| File | Severity | Location | Status |
|---|---|---|---|
| All | Informational | N/A | ACK |

**Description** - We recommend using a non-reentrant modifier in the claim functions to ensure no function can be called recursively.

## 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