



# SOLIDProof

*Bring trust into your projects*

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

MADE IN GERMANY

# Universal Chain

# AUDIT

SECURITY ASSESSMENT

**20. July, 2023**

FOR

 UNIVERSAL  
CHAINS



**SolidProof\_io**



**@solidproof\_io**

Introduction	3
Disclaimer	3
Project Overview	4
Summary	4
Social Medias	4
Audit Summary	5
File Overview	6
Imported packages	6
Components	7
Exposed Functions	7
StateVariables	7
Capabilities	8
Inheritance Graph	9
Audit Information	10
Vulnerability & Risk Level	10
Auditing Strategy and Techniques Applied	11
Methodology	11
Overall Security	12
Upgradeability	12
Ownership	13
Ownership Privileges	14
Minting tokens	14
Burning tokens	15
Blacklist addresses	16
Fees and Tax	17
Lock User Funds	18
Centralization Privileges	19
Audit Results	20



## Introduction

[SolidProof.io](https://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](https://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	Universal Chains
Website	<a href="https://www.universalchains.io/">https://www.universalchains.io/</a>
About the project	UniversalChains is a pioneering initiative with the goal of connecting users to different chains. Blockchain technology remains a resource of untapped potential. UniversalChains aims to increase growth and activity across networks, empowering others to engage & utilize different chains via the use of LayerZero technology and more.
Chain	BSC, Optimism, ZkSync, Polygon ZkEVM, Arbitrum, Arbitrum Nova, Avalanche
Language	Solidity
Codebase Link/ Address	<b>ZkSync:</b> 0x085d22740e40d15FdD754ae2C3B3E5FC7a17e12A <b>Optimism:</b> 0x76fFb8B75c37aC3C3849C8b3603aF8A1AA9245f5 <b>BSC:</b> 0xA756875ff1dEDFBee848896cbE86783ba5c939a1 <b>Polygon ZkEVM:</b> 0x950100D75f0308fBd0953f25C6b108238e11321f <b>Arbitrum:</b> 0xE925c62a17332F7f75B515B89e5919247514389C <b>Arbitrum Nova:</b> 0x950100D75f0308fBd0953f25C6b108238e11321f <b>Avalanche:</b> 0x1169226a38715A9eb1B776205E780C2f165DCe64
Unit Tests	Provided

## Social Medias

Telegram	N/A
Twitter	<a href="https://twitter.com/UniChains">https://twitter.com/UniChains</a>
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	N/A
Discord	<a href="https://discord.gg/Hjg9c7KgUS">https://discord.gg/Hjg9c7KgUS</a>

<b>Telegram</b>	N/A
<b>Youtube</b>	N/A
<b>TikTok</b>	N/A
<b>LinkedIn</b>	N/A

## Audit Summary

Version	Delivery Date	Changelog
v1.0	20. July 2023	<ul style="list-style-type: none"> <li>• Layout Project</li> <li>• Automated- /Manual-Security Testing</li> <li>• Summary</li> </ul>

**Note** - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. 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 it was not functionally tested by us.



## File Overview

The project 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/ UniversalChainsONFT721.sol	184d97795c80bb9c5520b99194523e222424c229

*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
@layerzerolabs/solidity-examples/contracts/token/onft/ONFT721.sol	1
@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.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'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

 <b>Contracts</b>	 <b>Libraries</b>	 <b>Interfaces</b>	 <b>Abstract</b>
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.

 <b>Public</b>	 <b>Payable</b>
10	2





<b>External</b>	<b>Internal</b>	<b>Private</b>	<b>Pure</b>	<b>View</b>
7	9	0	0	2

## StateVariables

<b>Total</b>	 <b>Public</b>
14	13



# Capabilities

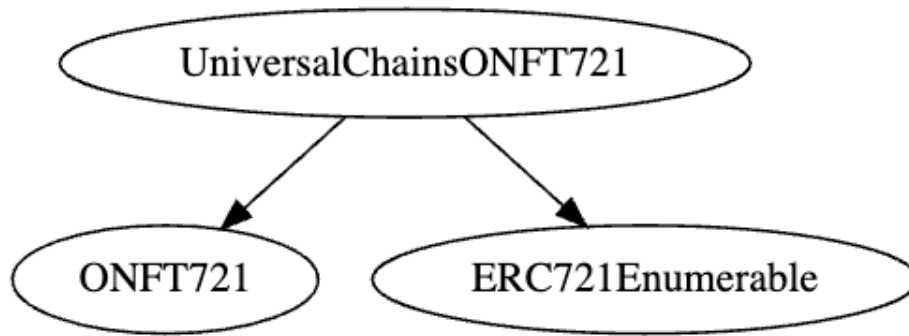
<b>Solidity Versions observed</b>	 <b>Transfers ETH</b>	 <b>Can Receive Funds</b>	 <b>Uses Assembl y</b>	 <b>Has Destroyable Contracts</b>
<code>^0.8.0</code>	Yes	Yes	---	----





## 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.



## 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)
<b>Critical</b>	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.
<b>High</b>	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 as possible.
<b>Medium</b>	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.
<b>Low</b>	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.
<b>Informational</b>	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:

- Centralizations
- The owner has significant control over contract's operations

Comment

N/A

## 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

The functionality is absent.





## 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 charge unfair fees/taxes**

Description	The contract levy a minting fee from all users and can be set to any arbitrary value
Example	Our assumption is that the owner can adjust the minting fee to a very high value or set it to a bare minimum value (even zero) for personal gain. Moreover, this will also result in the lock of minting functionality
Comment	We recommend setting a Hard Cap on minting fee

### Codebase:

```

120     function setMintingFee(uint256 _mintingFee↑) external onlyOwner {
121         uint256 oldMintingFee = mintingFee;
122         mintingFee = _mintingFee↑;
123         emit MintingFeeUpdated(oldMintingFee, _mintingFee↑);
124     }

```



## 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 contract**

**✗ The owner is able to lock the contract**

Description	Locking the contract means that the owner is able to lock any functionality in the contract so that the users are not able to use it anymore.
Example	An example of locking is by setting the minting fee to a very high value so that no one would be able to mint anymore
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
Main	<ul style="list-style-type: none"> <li>❖ <b>onlyOwner</b> <ul style="list-style-type: none"> <li>- Set Minting fee to any arbitrary value</li> <li>- Set Token URI</li> <li>- Lock Token URI</li> <li>- Set Protocol Address</li> </ul> </li> </ul>

## 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

## #1 | Functionality can be locked by the owner

File	Severity	Location	Status
Main	Medium	L120	Open

**Description** - The owner can lock the mint function by setting the minting fees to a really high value.

**Remediation** - Make sure that the fees can be set up to a maximum limit.

## #2 | Owner can Drain Protocol Earnings

File	Severity	Location	Status
Main	Medium	L128	Open

**Description** - The owner can set the protocol earning address as any address including their own which will let them claim all the protocol earnings

**Remediation** - Make sure that the protocol address is either a contract or the address is constant so that it cannot be changed.

## #3 | Floating Pragma

File	Severity	Location	Status
Main	Informational	L2	Open

### Description

- The current pragma Solidity directive is "`^0.8.0`". Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions.

## 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