



SOLIDProof

Bring trust into your projects

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

MADE IN GERMANY

Defi Pool Share

-
Lending

Audit

Security Assessment
26. July, 2023

For



DeFiPool
SHARE



SolidProof_io



@solidproof_io

Disclaimer	3
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Inheritance Graph	12
CallGraph	13
Scope of Work/Verify Claims	14
Modifiers and public functions	18
Source Units in Scope	19
Critical issues	20
High issues	20
Medium issues	20
Low issues	21
Informational issues	21
Audit Comments	22
SWC Attacks	23

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 Uniswap, 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 security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	15. June 2023	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary
1.1	26. July 2023	<ul style="list-style-type: none">• Reaudit

Network

Ethereum (ERC20)

Binance Smart Chain (BEP20)

Website

<https://defipoolshare.io/>

Telegram

<https://t.me/DefiPoolShare>

Twitter

<https://twitter.com/defipoolshare>

Discord

<https://discord.gg/7BWedF7Msx>



Description

TBA

Project Engagement

During the 12th of June 2023, **Defi Pool Share Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo



Contract Link

v1.0

- Provided as files

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. 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 as possible.
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 evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
 - ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.
2. Testing and automated analysis that includes the following:
 - i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

Dependency / Import Path	Count
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	1
@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol	1
@uniswap/v3-periphery/contracts/interfaces/INonfungiblePositionManager.sol	1



Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

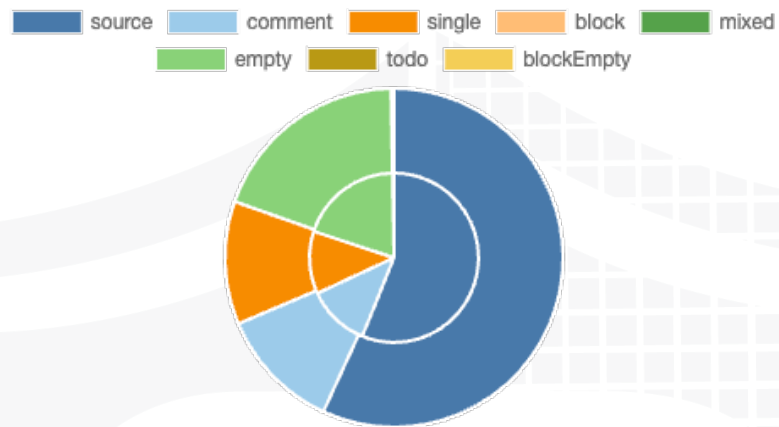
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

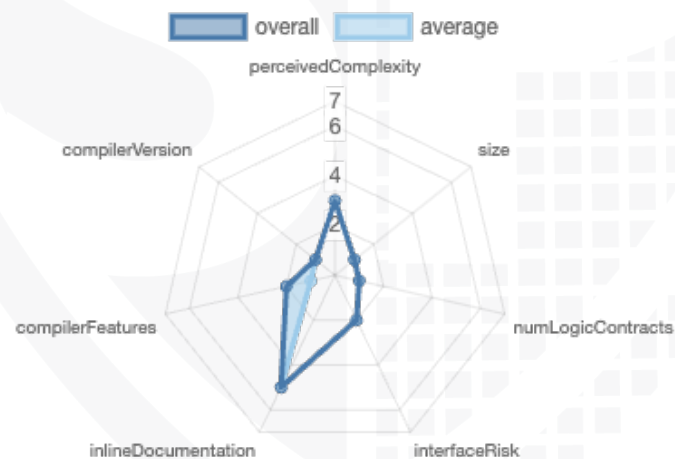
File Name	SHA-1 Hash
contracts/DPSLendingUniswapLiquidity.sol	ec5ccd1240a90ff43a860f96f2c2e1f463e563cb

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	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.

Version	Public	Payable
1.0	15	0

Version	External	Internal	Private	Pure	View
1.0	8	10	1	1	6

State Variables

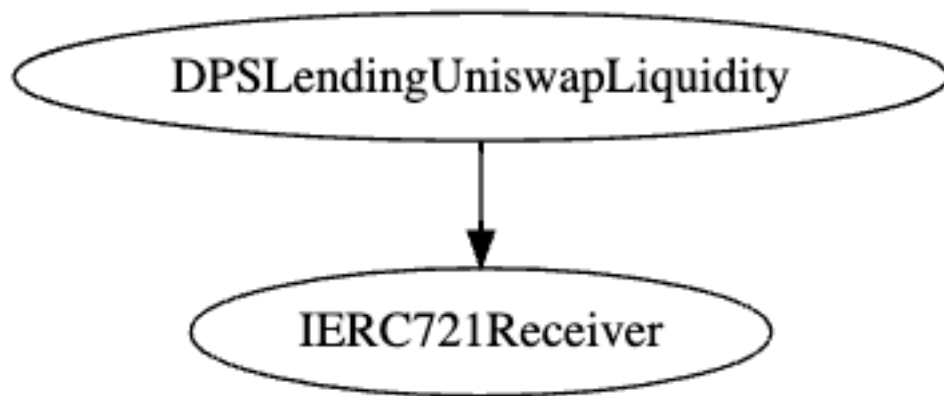
Version	Total	Public
1.0	10	10

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	^0.7.5				

Inheritance Graph

v1.0



CallGraph

v1.0



Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Is contract an upgradeable
2. Overall checkup (Smart Contract Security)

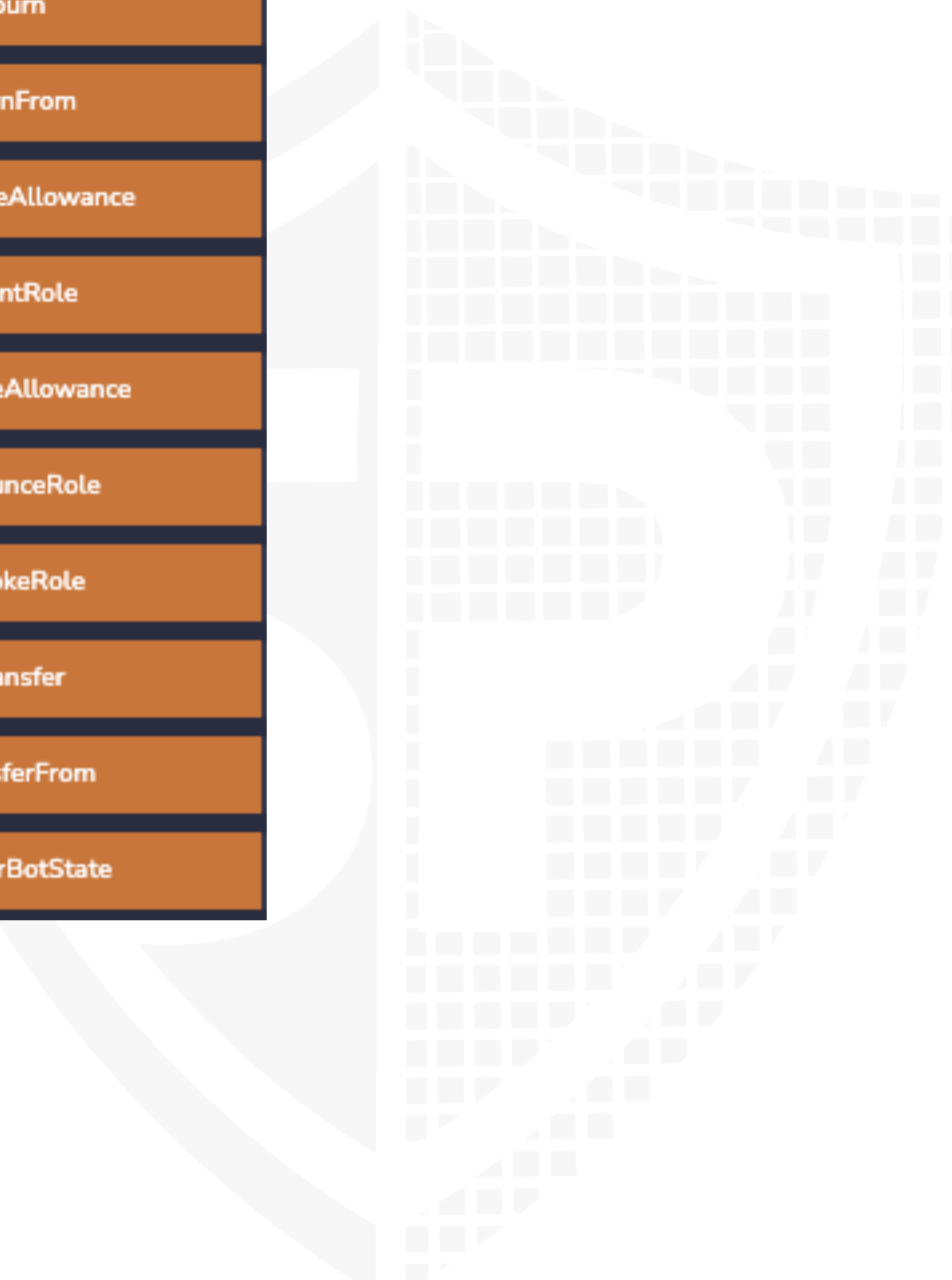


Is contract an upgradeable

Name	
Is contract an upgradeable?	No



Write functions of contract v1.0



approve
burn
burnFrom
decreaseAllowance
grantRole
increaseAllowance
renounceRole
revokeRole
transfer
transferFrom
triggerBotState

Overall checkup (Smart Contract Security)

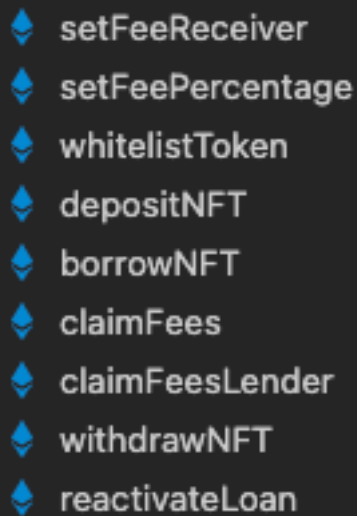
Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	🚩
Unverified / Not checked	✗
Not available	—

Modifiers and public functions

v1.0



- ◆ setFeeReceiver
- ◆ setFeePercentage
- ◆ whitelistToken
- ◆ depositNFT
- ◆ borrowNFT
- ◆ claimFees
- ◆ claimFeesLender
- ◆ withdrawNFT
- ◆ reactivateLoan

Comments

- There was not used a modifier directly but in the functions, there is a require statement that checks the caller is the owner's address. The following functions are using the require statement:
 - setFeeReceiver
 - SetFeePercentage
- The “whitelistedTokens” state variable is to whitelist all tokens to accept the token for the payment of the loan

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0

File Name	SHA-1 Hash
contracts/DPSLendingUniswapLiquidity.sol	ec5ccd1240a90ff43a860f96f2c2e1f463e563cb

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

Critical issues

No critical issues

High issues

High issues fixed

Issue	File	Type	Line	Description	Status
#1	Main	Unset _dpstDeployer	113	<p>The "_dpstDeployer" address is not set in the contract and also, it cannot be set. The "whitelist Token" functions checks the caller is _dpstDeployer*. In this case nobody is able to set a token as a whitelist.</p> <p>It is recommended to set the "_dpstDeployer" in the constructor.</p>	Fixed
#2	Main	Underflow/Overflow	See description	<p>Since the pragma version is below 0.8.x the underflow/overflow is not handled by default. This can cause problems in the contract.</p> <p>If the owner set the percentage higher than 100% and someone borrows a NFT the netLoanAmount L163 will be nearly the max uint256 because of the underflow.</p> <p>It is recommended to check every raw mathematical operations and replace them with SafeMath library operations.</p>	Fixed

Medium issues

No Medium issues found

Low issues

Issue	File	Type	Line	Description	Status
#1	Main	A floating pragma is set	2	Choosing a certain pragma version is recommended. It is preferred to use at least the 0.8.19 version Be aware of the underflow/overflow handling in the contract. Everything under 0.8.x must be handled manually e.g. with the SafeMath library	Open
#2	Main	Missing Zero Address Validation (missing-zero-check)	96, 84	Check that the address is not zero. If the feeReceiver is set to a zero/dead address, the fees will be lost.	Open
#3	Main	Claim Fees	194	The borrower can only claim the fees while the current timestamp is below the loan.endTime. Afterwards, the borrower is not able to claim his funds. Also, the borrower is able to call the claimFees as much as he wants in the period of the time.	Open
#4	Main	Missing Events Arithmetic	101	Emit an event for critical parameter changes.	Open
#5	Main	Fees can be set without a range	101	Fees can be set to 99%. Make sure to implement a range of 0-25% max.	Open

Informational issues

Issue	File	Type	Line	Description	Status
#1	Main	State variables that could be declared immutable	61, 52	Add the `immutable` attribute to state variables that never change or are set only in the constructor.	Open
#2	Main	TODO comment in the contract	92	Check the contract whether there is missing something. The TODO comment in the L93 indicates it.	Open

#3	Main	Zero value check	139	The “loanAmount” can be set to 0. Check the value is not 0 is recommended.	Open
#4	Main	Loan duration check	288	The loan duration was not checked whether it is above the current block.timestamp. Do it the same way as it was done in the “depositNFT” function.	Open
#5	Main	Stack too deep while compiling	239, 244	While compiling the contract the “Stack too deep” issue occurred. It is because of the positionManager.positions() functions. Here is how to solve this issue: https://medium.com/@sandnc.eth/uniswap-v3-stack-too-deep-error-message-d268dc8b57db	Open
#6	Main	Owner cannot be renounced	61	It is not possible to renounce the owner after deployment. We recommend to add functions to transfer/renounce the ownership.	Open
#7	Main	Tautology or contradiction	89, 103	Fix the incorrect comparison by changing the value type or the comparison. The “_feePercentage” is an uint256 type. This means that the value can never below 0.	Open

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information <https://docs.soliditylang.org/en/latest/natspec-format.html>) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

26. July 2023:

- There is still an owner (Owner still has not renounced ownership)
- Read whole report and modifiers section for more information

SWC Attacks

ID	Title	Relationships	Status
SW C-1 36	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-1 35	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-1 34	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-1 33	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-1 32	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-1 31	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-1 30	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-1 29	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-1 28	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-1 27	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-1 25	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-1 24	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-1 23	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-1 22	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-1 21	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-1 20	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SW C-11 9	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-11 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SW C-11 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 1	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-1 09	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-1 08	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-1 07	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-1 06	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-1 05	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-1 04	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-1 03	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	NOT PASSED
SW C-1 02	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-1 01	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	NOT PASSED
SW C-1 00	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED



*Solid
Proofed*

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


MADE IN GERMANY