SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

ZkSwap

Audit

Security Assessment

09. June, 2023

For

SolidProof_io

@solidproof_io

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 security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	06.June 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Note - This Audit report consists of a security analysis of the **ZkSwap** smart contracts. This analysis did not include functional testing (or unit testing) of the contract’s logic.

Network
zkSync

Website
https://zk-swap.xyz

Medium
https://medium.com/@_zkSwap

Twitter
https://twitter.com/_zkSwap

Discord
https://discord.com/invite/zk-swap

Description

The zkSwap team is dedicated to activating the dormant liquidity within DEXs by providing concentrated liquidity. An increase in liquidity utilization rate means more liquidity provided by the LPs will be utilized for the actual swaps, which in turn, provides bigger swap fees for the LPs. Taking a step further, we aim to increase the total transaction volume in the ecosystem by activating the overall liquidity in the zkSync ecosystem.

Project Engagement

During the 01 of June 2023, **ZkSwap 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.

Logo

Contract Links

v1.0

https://github.com/ZK-Swap-xyz/zk-swap-contracts/tree/main/contracts

Commit: f9a7a20

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 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 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.

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.

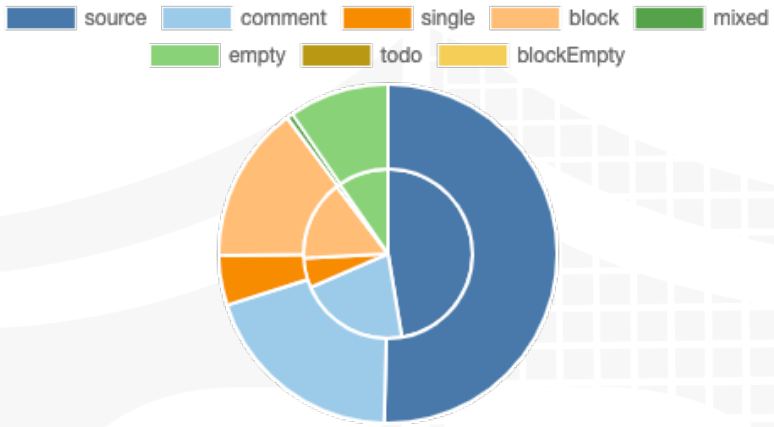
v1.0

File Name	SHA-1 Hash
contracts/periphery/ TicksFeesReader.sol	abaa86fc3a0ef95f15386b5c68e a9f4b9e4a206b
contracts/periphery/Quoter.sol	a7f9a1520c8ed6753749ce286b e4f9fce0c4af58
contracts/periphery/ TokenPositionDescriptor.sol	937bde775f1b627a8d5ce61981f 57624afa2de52
contracts/periphery/Router.sol	6ffad3daee2861472ddcb7e5d82 47c03310b13b7
contracts/periphery/ BasePositionManager.sol	a490b715b3312e300c4dbd8e6a 209b8291782399
contracts/periphery/ AntiSnipAttackPositionManager.sol	958b86bf130ddb527cda495b4a e8715d2d2ee743
contracts/PoolTicksState.sol	104035b0e185261bfa9729d9a1 252f8640f268cc
contracts/PoolStorage.sol	04851882aadc380cd2fc4376a0d 93dc97d0a84ef
contracts/Pool.sol	2d47dd63b7acc803693ac57739 b62e97f4a3bf94
contracts/Factory.sol	7dc485f6972d5065e1b978ef9a6 4a7ff717ee9fb

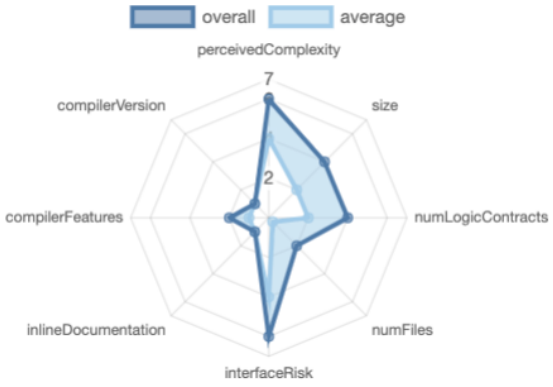
Note - Libraries, Base, Oracle, and Interfaces are excluded from the audit scope

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	💊Abstract
9	0	0	1

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
59	11

External	Internal	Private	Pure	View
46	72	11	2	30

StateVariables

Total	🌐Public
37	23

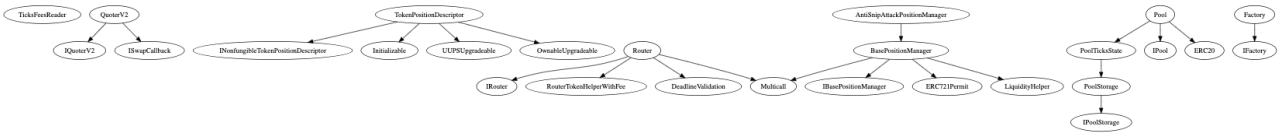
Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`0.8.16`		`yes`	`yes` (3 asm blocks)	

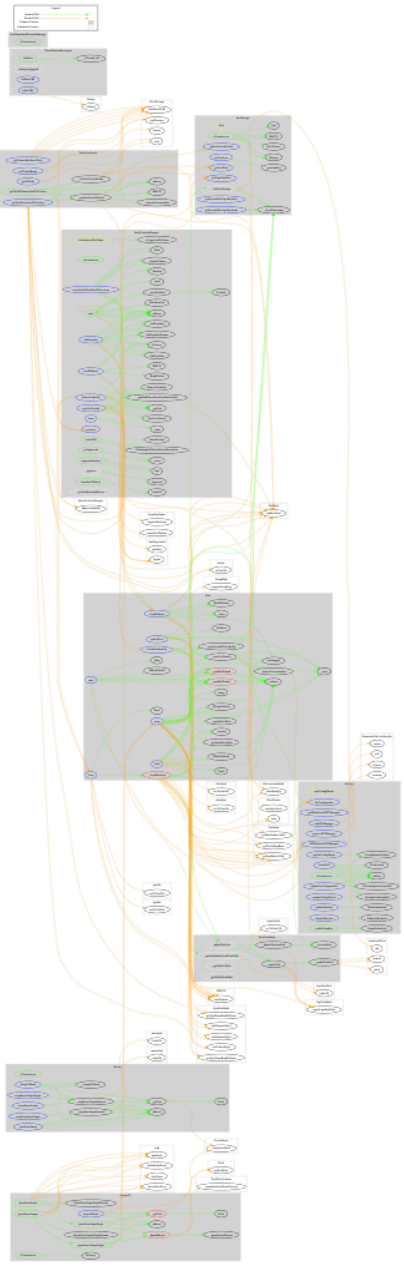
📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🦑 ECRecover	🌀 New/Create/Create2
			`yes`		

♻️ TryCatch	Σ Unchecked
`yes`	`yes`

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. Overall checkup (Smart Contract Security)

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

Note:

- ❖ This is a general 1:1 fork from Kyber Swap. Thus, all the contracts in the zkSwap codebase are identical to KyberSwap's contracts in terms of logic
    - Contracts inside are the same as the pancake-smart-contracts directory
        - https://github.com/KyberNetwork/ks-elastic-sc/tree/main/contracts
        - Differences between zkSwap and KyberSwap contracts are the following:
            - Only the pragma version is changed, and minor formatting has been done

Ownership/Admin Privileges

- ❖ Privileges are also identical to the KyberSwap contracts.

Note for Developers - We strongly advise the **zkSwap** team to analyse and modify the Native Token transfers in and out of the contracts because "zkSync" has no native currency, so functions, where the native currency is transferred, will not work.

Source Units in Scope
v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/periphery/TicksFeesReader.sol	1		217	189	147	25	108
contracts/periphery/Quoter.sol	1		264	217	172	24	211
contracts/periphery/TokenPositionDescriptor.sol	1		39	34	27	1	25
contracts/periphery/Router.sol	1		231	189	141	19	108
contracts/periphery/BasePositionManager.sol	1		376	293	229	16	137
contracts/periphery/AntiSnipAttackPositionManager.sol	1		182	146	118	4	56
contracts/PoolTicksState.sol	1		266	224	167	37	41
contracts/PoolStorage.sol	1		229	192	140	22	50
contracts/Pool.sol	1		666	602	457	98	231
contracts/Factory.sol	1		200	183	115	34	95
Totals	**10**		**2670**	**2269**	**1713**	**280**	**1062**

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

No high issues

Medium issues

Issue	File	Type	Line	Description	Status
#1	AntiSnipAttcakPositionManager.sol BasePositionManager.sol Router.sol	Native Transfers are Unsupported	—	All the contracts that have payable functions (Native Token Transfers) will not work. For example, statements used in EVM based smart contracts like “receipient.transfer(amount)”, the contracts should use their own methods for transfer rather than using methods like “call”. Moreover, in every transaction the token which is being transferred must be declared explicitly.	Fixed

Low issues

No low issues

Informational issues

Issue	File	Type	Line	Description
#1	All	Contract doesn’t import npm packages from source (like OpenZeppelin etc.)	—	We recommend importing all packages from npm directly without flattening the contract. Functions could be modified or can be susceptible to vulnerabilities

Audit Comments

We recommend you use the particular form of comments (NatSpec Format, Follow the 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 those variables, functions etc. do.

09. June 2023:

- This project consists of the following forks
  - **KyberSwap**
- Unit tests with at least 95% code coverage and a Whitepaper were not provided to SolidProof, so we cannot ensure the complete functional correctness of the code's logic.
- We recommend **zkSwap** team conduct unit and fuzz tests thoroughly to rule out the possibilities of unwanted logical and calculation errors.
- Read the whole report and modifiers section for more information
- The low issues that remain unfixed in the **Kyber Swap** codebase still exist in the forked code.
- We recommend using a multi-sig wallet for the owner's address to prevent any risk of the loss of a private key.
- Do your own research here before investing.

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED