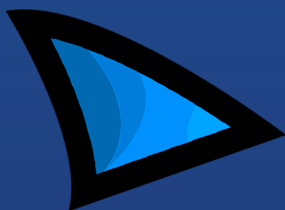# SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Sharky Swap
Factory/Pair
Audit

Security Assessment
21. February, 2023

For

SharkySwap

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	21. February 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Arbitrum One

Website
https://sharkyswap.xyz/

Twitter
https://twitter.com/SharkySwapFi

Github
https://github.com/sharkyswap/

Medium
https://sharkyswap.medium.com/

Discord
https://discord.gg/sharkyswap

Description

TBA

Project Engagement

During the 18th of February 2023, **SharkySwap 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

- https://arbiscan.io/address/ 0x36800286f652dDC9bDcFfEDc4e71FDd207C1d07C#code
- https://arbiscan.io/address/ 0x8a984f6bcaec0dc0a48d2435bbf76eb6ebf1c56e
- https://arbiscan.io/address/ 0x5c3d7b1e28a2bc612e9c92777eb144c1be3151e7
- https://arbiscan.io/address/ 0x36f3AEeF648641249fa35697134cEdedee6DdAeC
- https://arbiscan.io/address/ 0x41EE01d49cC01751B7E1DB92d82aA54e7fc133A7
- https://arbiscan.io/address/ 0x2721f1946Dd86FEF97c7499456F116aBa7f60f3F

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.
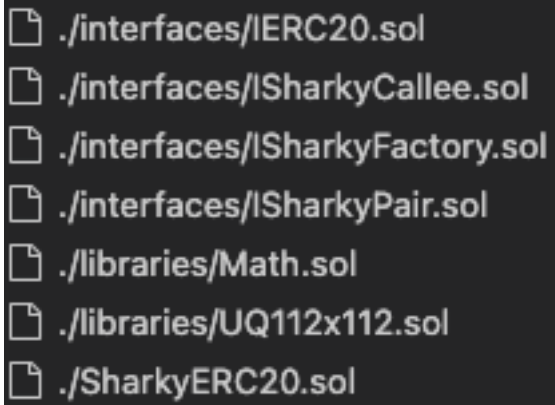
Methodology

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.

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

Imported packages:

```
📄 ./interfaces/IERC20.sol
📄 ./interfaces/ISharkyCallee.sol
📄 ./interfaces/ISharkyFactory.sol
📄 ./interfaces/ISharkyPair.sol
📄 ./libraries/Math.sol
📄 ./libraries/UQ112x112.sol
📄 ./SharkyERC20.sol
```

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/sources/interfaces/ISharkyCallee.sol	e246ef12320a94f8ee5a6dd4813af77cf8c49f0c
contracts/sources/interfaces/ISharkyFactory.sol	02c54e4bbfbbc805ab80ca537414fd48628bb725
contracts/sources/interfaces/ISharkyERC20.sol	ae8b24482aae0ca98f0360f3ed60005c5f388268
contracts/sources/interfaces/ISharkyPair.sol	7239b48c2ecad04c458a4da3192fec17d1c51192
contracts/sources/SharkyPair.sol	29b0e7ef441883b8a800e8336fcaff7761094914
contracts/sources/libraries/Math.sol	99c7ffd6351c1a3e7c5e8736313f7bea4abd989a
contracts/sources/libraries/UQ112x112.sol	97350dc58734f00336bb7f4bade4e715c214a9de
contracts/sources/SharkyFactory.sol	4fea5de769b533065f45458a9ebc5d9f4881b4df

Metrics

Source Lines
v1.0
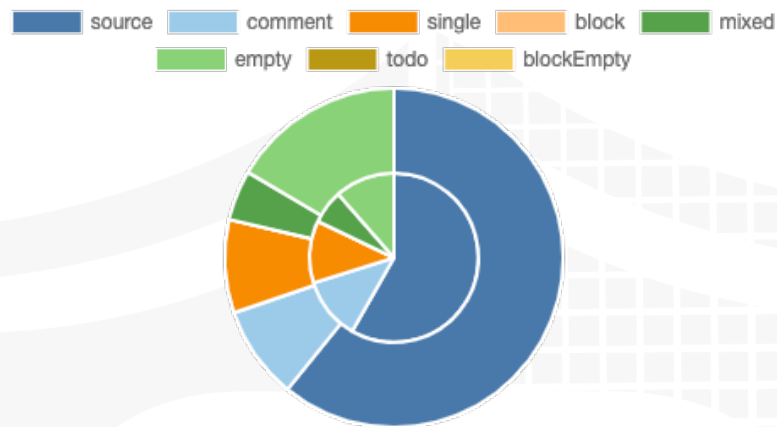
Risk Level
v1.0
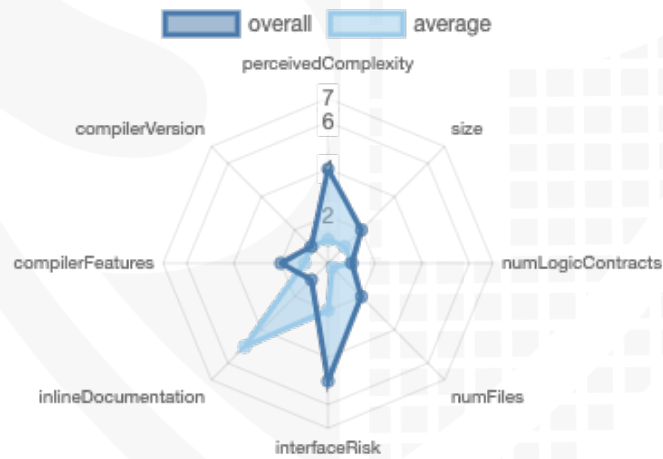
Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	2	4	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	63	0

Version	External	Internal	Private	Pure	View
1.0	60	36	3	13	25

State Variables
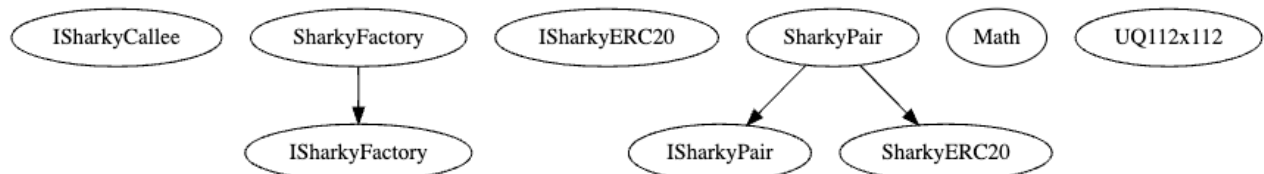
Version	Total	Public
1.0	18	12

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	>=0.5.0 =0.5.16			yes (1 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/ Create/ Create2
1.0				yes		yes → AssemblyCall:Name:create2

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

1. createPair

2. setFeeTo

3. setFeeToSetter

1. approve

2. burn

3. initialize

4. mint

5. permit

6. skim

7. swap

8. sync

9. transfer

10. transferFrom

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

Factory

createPair
setFeeTo
setFeeToSetter

Pair

initialize
mint
  lock
burn
  lock
swap
  lock
skim
  lock
sync
  lock

approve
transfer
transferFrom
permit

Comments

Note:
- feeToSetter is the owner in this case
- Following contracts are a fork of pancake swap
  - SharkyPair
  - SharkyERC20
  - SharkyFactory
  - Libraries
  - interfaces

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

Source Units in Scope
v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/sources/interfaces/ISharkyCallee.sol	——	1	11	5	3	1	3	——
🔍	contracts/sources/interfaces/ISharkyFactory.sol	——	1	24	7	4	1	19	——
🔍	contracts/sources/interfaces/ISharkyERC20.sol	——	1	45	8	5	1	27	——
🔍	contracts/sources/interfaces/ISharkyPair.sol	——	1	97	8	5	1	55	——
📝	contracts/sources/SharkyPair.sol	1	——	202	202	167	35	186	▨
📚	contracts/sources/libraries/Math.sol	1	——	24	24	18	3	5	——
📚	contracts/sources/libraries/UQ112x112.sol	1	——	21	21	10	7	4	——
📝	contracts/sources/SharkyFactory.sol	1	——	53	53	42	3	53	🖥️📇🌀
📝📚🔍	**Totals**	4	4	477	328	254	52	352	🖥️📇🌀

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

<div style="background-color:green;color:green;text-align:center">No critical issues</div>

High issues

<div style="background-color:green;color:green;text-align:center">No high issues</div>

Medium issues

<div style="background-color:green;color:green;text-align:center">No medium issues</div>

Low issues

Issue	File	Type	Line	Description
#1	SharkFactory	Missing Zero Address Validation (missing-zero-check)	18, 44, 49	Check that the address is not zero
#2	SharkyPair	Missing Zero Address Validation (missing-zero-check)	67	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	Interfaces/ libraries	Old pragma version + floating pragma	See description	We recommend you to use a newer pragma version for the contracts. Use at least the pragma version above 0.8.x.

Commented Code exist
There are some instances of code being commented out in the following files that should be removed:

File	Line	Comment
SharkyERC20	18	// keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");

Recommendation

Remove the commented code, or address them properly.

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.

21. February 2023:

· Read whole report and modifiers section for more information

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](#)	**NOT 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**

MADE IN GERMANY

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing