SMART CONTRACTS REVIEW

zokyo

May 28th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING SHIDO

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 1 Critical issue: 1 resolved = 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 0 Low issues: 0 points deducted
- 2 Informational issues: 2 resolved = 0 points deducted

Thus, the score is 100.

TECHNICAL SUMMARY

This document outlines the overall security of the Shido smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Shido smart contract/s codebase for quality, security, and correctness.

Contract Status

There was 1 critical issue found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Shido team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Shido repository:
Repo:
- https://github.com/ShidoGlobal/shido-liquidity-pool
- https://github.com/ShidoGlobal/shido-liquidity-pool-core
Fixes: https://github.com/ShidoGlobal/wSHIDO/blob/main/contracts/WSHIDO.sol

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ./UniswapV3PoolDeployer.sol
- ./libraries/Position.sol
- ./libraries/LiquidityMath.sol
- ./libraries/TickBitmap.sol
- ./libraries/Tick.sol
- ./libraries/BitMath.sol
- ./libraries/LowGasSafeMath.sol
- ./libraries/Oracle.sol
- ./libraries/TickMath.sol
- ./libraries/FixedPoint128.sol
- ./libraries/SqrtPriceMath.sol
- ./libraries/TransferHelper.sol
- ./libraries/FullMath.sol
- ./libraries/SafeCast.sol
- ./libraries/FixedPoint96.sol
- ./libraries/UnsafeMath.sol
- ./libraries/SwapMath.sol
- ./NoDelegateCall.sol
- ./UniswapV3Factory.sol
- ./UniswapV3Pool.sol
- v3-periphery-main
- ./V3Migrator.sol
- ./libraries/SqrtPriceMathPartial.sol
- ./libraries/LiquidityAmounts.sol
- ./libraries/HexStrings.sol
- ./libraries/Path.sol
- ./libraries/PositionKey.sol
- ./libraries/PoolAddress.sol
- ./libraries/PositionValue.sol
- ./libraries/ChainId.sol
- ./libraries/NFTSVG.sol
- ./libraries/TokenRatioSortOrder.sol
- ./libraries/PoolTicksCounter.sol
- ./libraries/CallbackValidation.sol
- ./libraries/TransferHelper.sol
- ./libraries/OracleLibrary.sol
- ./libraries/NFTDescriptor.sol
- ./libraries/BytesLib.sol
- ./SwapRouter.sol
- ./NonfungiblePositionManager.sol
- ./NonfungibleTokenPositionDescriptor.sol
- ./base/SelfPermit.sol
- ./base/LiquidityManagement.sol
- ./base/PoolInitializer.sol
- ./base/PeripheryImmutableState.sol
- ./base/PeripheryPaymentsWithFee.sol
- ./base/PeripheryPayments.sol
- ./base/ERC721Permit.sol
- ./base/Multicall.sol
- ./base/BlockTimestamp.sol
- ./base/PeripheryValidation.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Shido smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security review of the provided codebase. The contracts submitted for auditing are well-crafted and well organized with a key design choice of forking their contracts from Uniswap and the Wrapped ETH contract for WSHIDO tokens. Detailed findings from the audit process are outlined in the "Complete Analysis" section.

The Shido codebase for the audit consists of contracts that make heavy use of the Uniswap V3 (apart from some changes to display strings), the Wrapped ETH contract which minimal changes in order to adapt to the modern Solidity compiler, and some custom contracts to represent Tether (USDT) and Aave tokens. With the changes to the Solidity compiler throughout the years, relevant recommendations outlined in this report were made in order to mitigate critical and medium severity vulnerability classes. As the Uniswap V3 codebase is tried and tested by withstanding the test of time, more of the team's attention was given to where most of the changes had taken place - the WSHIDO and custom ERC20 contracts. Following the engagement, the fixes were tested again against proof of concept exploit code to confirm that these edge cases were not possible.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Shido team and the Shido team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Insufficient Src Validation allows Malicious Users To Steal WSHIDO From Users With Zero Allowance	Critical	Resolved
2	Deprecated Native Token Transfer Method Used In Withdraw May Cause Reverts	Medium	Resolved
3	Use Constant and Immutable to reduce smart contracts gas cost	Informational	Resolved
4	Addition of Reentrancy Guards	Informational	Resolved

CRITICAL-1 | RESOLVED

Insufficient Src Validation allows Malicious Users To Steal WSHIDO From Users With Zero Allowance

The WSHIDO token is forked from the WETH contract on the Ethereum Mainnet and is intended to allow users to wrap SHIDO to be used in the ecosystem. In the `transferFrom` function, there is a check which ensures that if `src` address is not equal to the `msg.sender` and the allowance is not equal to zero, deduct from allowance. After this check with no conditions, deduct from the `src` address and add to the `dst` address which both variables are user supplied and is where the issue lies. As a result, free token transfers can be made from users with zero allowance.

Recommendation:
It's recommended that the `transferFrom` method for the WSHIDO contract is refactored to include logic to account for the `src` address being the `msg.sender` and deduct amounts accordingly.

MEDIUM-1 | RESOLVED

Deprecated Native Token Transfer Method Used In Withdraw May Cause Reverts

The `withdraw()` method is used by the `WSHIDO` contract in order to unwrap native tokens and which uses `payable(*).transfer(wad)` to transfer the originally deposited amount to the user. The original transfer method uses a fixed stipend of 2,300 gas units which may not be sufficient for some contracts to process the transfer resulting in a revert, preventing the receiving of funds.

Recommendation:
It's recommended that a low level `.call()` is used to transfer Ether between contracts and EOAs.

INFORMATIONAL-1 | RESOLVED

Use Constant and Immutable to reduce smart contracts gas cost

The WSHIDO smart contract utilizes state variables `name`, `symbol`, and decimals which are initialized at deployment and remain unchanged thereafter. These variables are currently declared without the `constant` keyword, resulting in unnecessary gas costs when these variables are accessed during transactions.

Recommendation:
Convert the `name`, `symbol`, and `decimals` variables to constants by declaring them with the `constant` keyword. This change will embed these variables into the bytecode, eliminating the need for storage operations and reducing gas costs associated with accessing these values.

INFORMATIONAL-2 | RESOLVED

Addition of Reentrancy Guards

Functions such as withdraw are susceptible to reentrancy attacks, where recursive calls can lead to unexpected behaviors and potentially harmful consequences. These functions also tend to consume more gas due to the nature of such attacks.

Recommendation:
Implement reentrancy guards to minimize the risks associated with reentrant calls. In addition to this, reentrancy guards will also assist in managing the gas costs by preventing multiple unwarranted executions of expensive state changes.

Hacker's Notes (Informational)

The WSHIDO has been forked directly from the WETH code deployed on the Ethereum mainnet to cater to Shido's requirement of wrapping tokens. Developers should exercise caution when attempting to fork code from older code bases as there may have been breaking changes in the relative technologies since the code was written, for instance, the over/underflow checks added to the solidity compiler post 0.8 updates which required changes to the approval clause in `transferFrom` function. For this reason, we recommend that the WSHIDO code is refactored to suit the current landscape of Defi and implement ERC20 with added `receive(), deposit()`, and `withdraw()` functions. In addition to this, further changes to the `totalSupply` function will have to be made to consider the current balance of native tokens within the contract. For this reason, the token will fully comply with the ERC20 standards in addition to the gas optimizations made within the inherited contract. In addition to this, it may help with code size reduction in the main contract maintaining readability of the code.

Finally, the development team is to be commended for forking the liquidity pool source code directly from Uniswap V3 with minimal changes as the Uniswap V3 source code is a proven, battle tested code base which has withstood the test of time in the ever changing landscape of Defi and it's threats.

Post Fix Hacker's Notes

The updates to the WSHIDO contract seemed to have prevented the exploit to steal WSHIDO from other users with a zero allowance. It can also be noted that an initial owner was added as a constructor variable. The developers should be made aware as (informational ie. no security implications) that this initial owner value is somewhat redundant as there are no special powers or settings that the owner can do compared to other users.

	Contracts
Reentrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Shido team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Shido team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo