Security Assessment

Shield SS-Vault

Sep 5th, 2022

Secure3

secure3.io

Table of Contents

Table of Contents 1
Summary 2
Overview 3
Audit Scope 4
Code Assessment Findings 5
SLD-1: SSVault.dexType uninitialized value risk 7
SLD-2: SSVault reentrancy risk 8
SLD-3: SSVault::sendSettleRewards() should use oracle to get SLD token and gas price 10
SLD-4: SSVault::settleOrders() should check if orders are valid 11
SLD-5: SSVault.tokenAggregator is initialized but never used 12
SLD-6: SSVaultManager.meanPricePeriod 13
inconsistent naming convention for constant 13
SLD-7: SSVaultManager should use indexed keyword in events 14
SLD-8: ShieldOracleV2::consult() unused secondsAgo parameter 15
SLD-9: ShieldOracleV2::update() missing event 16
Disclaimer 17

Summary

Shield SS-Vault is a single-side staking vault that enables the vault issuers to customize on-chain covered call strategies for on-chain long-tail assets. Traditionally, the yield farming is rewarding based on the token quantity but not token price. SS-Vault innovative two APYs synthetic product provides more attractive incentive model, which allows users to get more return for underlying's token price upside while still preserves the return for token price downside.

This report has been prepared for the project to identify issues and vulnerabilities in the smart contract source code. A comprehensive examination with Static Analysis and Manual Review techniques has been performed.

The examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static scanner to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Informational, Medium, Critical. For each of the findings we have provided recommendation of a fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Shield SS-Vault
Platform & Language	Ethereum, Solidity
Codebase	repository - https://github.com/shielddeveloper/shield-vaults-v1 audit commit - 6b050e58e47890ef34de6b1b289af917b7f6470e final commit - 901f9ed551d8cb6a987a37ea2d900a96bc04393b
Audit Methodology	• Business Logic Understanding and Review • Static Analysis • Code Review

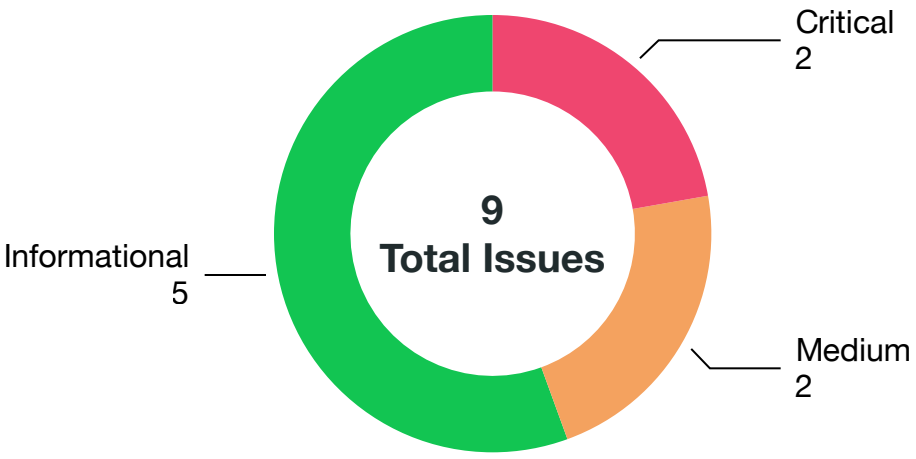
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowleged	Fixed	Mitigated
Critical	2	0	1	1	0
Medium	2	0	0	2	0
Informational	5	0	2	3	0

Audit Scope

File	Commit Hash
ssvault/ShieldOracleManager.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
ssvault/SSVault.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
ssvault/ShieldOracle.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
ssvault/SSVaultManager.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
ssvault/ShieldOracleV2.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/UniswapV2OracleLibrary.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/FixedPoint.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/SafeMath.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/TickMath.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/TransferHelper.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/FullMath.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
libraries/OracleLibrary.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
interfaces/IUniswapV2Factory.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
interfaces/IUniswapV2Pair.sol	6b050e58e47890ef34de6b1b289af917b7f6470e
interfaces/IBEP20.sol	6b050e58e47890ef34de6b1b289af917b7f6470e

Code Assessment Findings



ID	Name	Category	Severity	Status
SLD-1	SSVault.dexType uninitialized value risk	Code Style	Informational	Fixed
SLD-2	SSVault reentrancy risk	Logical	Critical	Fixed
SLD-3	SSVault::sendSettleRewards() should use oracle to get SLD token and gas price	Logical	Critical	Acknowledged
SLD-4	SSVault::settleOrders() should check if orders are valid	Logical	Medium	Fixed
SLD-5	SSVault.tokenAggregator is initialized but never used	Logical	Informational	Acknowledged
SLD-6	SSVaultManager.meanPricePeriod inconsistent naming convention for constant	Code Style	Informational	Fixed

ID	Name	Category	Severity	Status
SLD-7	`SSVaultManager` should use `indexed` keyword in events	Language Specific	Informational	Acknowledged
SLD-8	`ShieldOracleV2::consult()` unused `secondsAgo` parameter	Logical	Medium	Fixed
SLD-9	`ShieldOracleV2::update()` missing event	Logical	Informational	Fixed

SLD-1: `SSVault.dexType` uninitialized value risk

Category	Severity	Code Reference	Status
Code Style	Informational	contracts/ssvault/SSVault.sol:59,496	Fixed

Code

```
59:     uint256 public dexType; // 0 V2, 1 V3


496:         if (dexType == 0) {
497:             oracle.update();
498:         } else {
499:             require(dexType == 1 || dexType == 2, "wrong dex type");
500:             require(block.timestamp > endTime, "not end");
501:         }
```

Description

The `SSVault.dexType` is used to denote the DEX to be v2 (value 0) type or v3 (value 1, 2) type. However, value 0 can also be uninitialized default value of `uint256` type, hence it is recommended to use 0 as the undefined DEX type and 1, 2, 3 for the actual DEX type values.

Recommendation

Use Enums to define the types and reserve the value 0 to be the `undefined` DEX type.

Client Response

Fixed. The `DexType` enum has been added with 0 as the `INVALID` type.

SLD-2: `SSVault` reentrancy risk

Category	Severity	Code Reference	Status
Logical	Critical	contracts/ssvault/SSVault.sol	Fixed

Code

```
180:     function deposit(uint256 _amount) public notTerminated {
181:         require(_amount >= minDeposit, "deposit amount too small");
182:         require(block.timestamp < endTime, "vault ended");
183:         require(totalDeposit.add(_amount) <= maxVolume, "exceed vault volume");
184:
185:         UserInfo storage info = userInfo[msg.sender];
186:
187:         require(
188:             info.totalDeposit.add(_amount) <= maxDeposit,
189:             "exceed individual deposit"
190:         );
191:
192:         uint256 _before = baseToken.balanceOf(address(this));
193:         _safeTransferFrom(
194:             address(baseToken),
195:             msg.sender,
196:             address(this),
197:             _amount
198:         );
199:         uint256 _after = baseToken.balanceOf(address(this));
200:         // Additional check for deflationary tokens
201:         _amount = _after.sub(_before);
```

Description

The function `deposit()`, `withdraw()`, `payMargin()`, `paySLDMargin()`, `withdrawMargin()` in the `SSVault` has risk of reentrancy because of ERC777 token call back function. Taking `deposit()` function for example, `address(baseToken)` can be ERC777 token and if the contract has registered ERC1820 for callback function, `ERC777._callTokensToSend()` will be called and malicious code can call `deposit()` again to re-enter. Because `_amount` value is updated after the `_safeTransferFrom()` call, this can lead to more `_amount` to be deposited than the `maxVolume`.

Recommendation

Use **Checks-Effects-Interaction** pattern in the code and `ReentrancyGuard.sol` to prevent the possible reentrancy attacks.

Client Response

Fixed. The `nonReentrant` modifier has been used in the contract.

SLD-3: `SSVault::sendSettleRewards()` should use oracle to get SLD token and gas price

Category	Severity	Code Reference	Status
Logical	Critical	contracts/ssvault/SSVault.sol:397,403	Acknowledged

Code

```
395:     function sendSettleRewards(uint256 _gasUsed) internal {
396:         uint256 gasFeeUsed = _gasUsed
397:             .mul(gasPrice)
398:             .mul(getBNBPrice())
399:             .mul(feebackNumerator)
400:             .div(feebackDenominator)
401:             .div(MULTIPLIER);
402:
403:         uint256 rewards = gasFeeUsed.mul(sldPriceForRewardsDenominator).div(
404:             sldPriceForRewardsNumerator
405:         );
406:
407:         if (rewards > 0) {
408:             sldMargin = sldMargin.sub(rewards);
409:             _safeTransfer(address(sldToken), msg.sender, rewards);
410:         }
411:     }
```

Description

The function `sendSettleRewards()` sends `sldToken` at the end based on gas consumed to `msg.sender` to encourage users to actively maintain the contract state. However, the gas and SLD token prices are maintained by the contract owner as the state variable. If the `gasPrice` and `sldPriceForRewardsNumerator` token price are not updated in time and are far from what the current market price is, the attacker can repeatedly call `SSVault::updatePrice()` to obtain SldTokens and sell them to complete the arbitrage.

Recommendation

Use oracle to get the update to date SLD token and gas price.

Client Response

Acknowledged. The SLD token price is adjusted by the platform operation to control the incentive SLD token amount.

SLD-4: `SSVault::settleOrders()` should check if orders are valid

Category	Severity	Code Reference	Status
Logical	Medium	contracts/ssvault/SSVault.sol:387	Fixed

Code

```
379:     function settleOrders(uint256[] memory orderIDs) public {
380:         uint256 gasUsed;
381:
382:         require(isStriked, "not strike");
383:         require(settledPrice > 0, "need settle price");
384:         require(!isAllSettled(), "all settled");
385:
386:         for (uint256 i = 0; i < orderIDs.length; i++) {
387:             gasUsed = gasUsed + settleOrder(orderIDs[i]);
388:         }
```

Description

The `orderIDs[i]` value can potentially point to an invalid order.

Recommendation

Verify the index is valid by `orderIDs[i] < orders.length` check.

Client Response

Fixed. The check has been added.

SLD-5: `SSVault.tokenAggregator` is initialized but never used

Category	Severity	Code Reference	Status
Logical	Informational	contracts/ssvault/SSVault.sol:26	Acknowledged

Code

```
26:     address public tokenAggregator;
```

Description

State variable `tokenAggregator` is initialized in the constructor but never used anywhere else.

Recommendation

Check if `tokenAggregator` is indeed needed.

Client Response

Acknowledged. The `tokenAggregator` is used to provide price for frontend dApp.

SLD-6: `SSVaultManager.meanPricePeriod` inconsistent naming convention for constant

Category	Severity	Code Reference	Status
Code Style	Informational	contracts/ssvault/SSVaultManager.sol:16	Fixed

Code

```
10: contract SSVaultManager {
11:     using SafeMath for uint256;
12:
13:     uint256 internal constant MULTIPLIER = 1e18;
14:     uint256 internal constant DECIMALS = 18;
15:     uint256 internal constant SECONDS_IN_YEAR = 365 days;
16:     uint256 internal constant meanPricePeriod = 30 minutes;
```

Description

It is recommended to have consistent naming convention across variables.

Recommendation

Rename `meanPricePeriod` to `MEAN_PRICE_PERIOD` for `constant` naming consistency.

Client Response

Fixed.

SLD-7: `SSVaultManager` should use `indexed` keyword in events

Category	Severity	Code Reference	Status
Language Specific	Informational	contracts/ssvault/SSVaultManager.sol:53-72	Acknowledged

Code

```
53:     event VaultCreated(
54:         address vault,
55:         address holder,
56:         address token0,
57:         address token1,
58:         uint256 maxVolume,
59:         uint256 timestamp
60:     );
61:     event OracleCreated(address vault, address oracle);
62:
63:     event SetMarginRatio(uint256 oldValue, uint256 newValue);
64:     event SetMinVolume(uint256 oldValue, uint256 newValue);
65:     event SetMinPeriod(uint256 oldValue, uint256 newValue);
66:     event SetGovernance(address oldValue, address newValue);
67:
68:     event SetFeeRatio(address vault, uint256 value);
69:     event SetGasPrice(address vault, uint256 value);
70:     event SetSLDPrice(address vault, uint256 value0, uint256 value1);
71:     event SetParameters(address vault, uint256 value0, uint256 value1);
72:     event SetOracle(address vault, address value);
```

Description

Index the parameters in the defined event such as `event VaultCreated` to help easier consuming the contract on-chain data by filtering the desired indexed field.

Recommendation

Add `indexed` keywords to important parameter fields.

Client Response

Acknowledged. Keyword `indexed` is not added in this version for backward compatibility in the backend.

SLD-8: `ShieldOracleV2::consult()` unused `secondsAgo` parameter

Category	Severity	Code Reference	Status
Logical	Medium	contracts/ssvault/ShieldOracleV2.sol:86	Fixed

Code

```
86:     function consult(address tokenIn, uint32 secondsAgo)
```

Description

The `secondsAgo` parameter is not honored in the `consult()` function implementation but in the `SSVault.getTokenValue()`, parameter `60` is passed in. From the interface abstraction perspective, all the child contracts should implement `ShieldOracle` interface features, and `secondsAgo` represents the valid time frame of the price is valid for, hence `ShieldOracleV2` should also implement it. From security perspective, enforce the price valid time can prevent the stale price to be used accidentally for contract calculation.

Recommendation

Add a check comparing with `blockTimestampLast`, which is updated in the `ShieldOracle.update()` function. And require the `block.timestamp` is within `secondsAgo`.

Client Response

Fixed. The check for `secondsAgo` is added.

SLD-9: `ShieldOracleV2::update()` missing event

Category	Severity	Code Reference	Status
Logical	Information	contracts/ssvault/ShieldOracleV2.sol:44	Fixed

Code

```
44:     function update() external override {
```

Description

The price of the oracle will be updated periodically by calling `ShieldOracleV2::update()` externally. Without proper logging and monitoring, it could result the price being stale for a long time until the team gets noticed.

Recommendation

Emit an event with price as parameter at the end of the `update()` function and having a monitoring process externally to make sure the price is always up to date.

Client Response

Fixed. The `Update` event is added and emitted at the end of the function.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.