



# SMART CONTRACT AUDIT REPORT

for

## Hyperswap V2



Prepared By: Xiaomi Huang

PeckShield  
February 4, 2025

## Document Properties

Client	Hyperswap
Title	Smart Contract Audit Report
Target	Hyperswap V2
Version	1.0
Author	Xuxian Jiang
Auditors	Daisy Cao, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

## Version Info

Version	Date	Author(s)	Description
1.0-rc	January 11, 2025	Xuxian Jiang	Release Candidate #1

## Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Email	contact@peckshield.com

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	About Hyperswap V2 . . . . .	4
1.2	About PeckShield . . . . .	5
1.3	Methodology . . . . .	5
1.4	Disclaimer . . . . .	7
<b>2</b>	<b>Findings</b>	<b>9</b>
2.1	Summary . . . . .	9
2.2	Key Findings . . . . .	10
<b>3</b>	<b>Detailed Results</b>	<b>11</b>
3.1	Improved setStableSwap() Logic in HyperswapPair . . . . .	11
3.2	Implicit Assumption Enforcement In AddLiquidity() . . . . .	12
3.3	Improper LP Volume Accounting in HyperswapRouter . . . . .	14
3.4	Lack of Slippage Control in MoneyMaker . . . . .	15
3.5	Trust Issue of Admin Keys . . . . .	16
<b>4</b>	<b>Conclusion</b>	<b>18</b>
	<b>References</b>	<b>19</b>

# 1 | Introduction

Given the opportunity to review the design document and related source code of the `Hyperswap V2` protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

## 1.1 About Hyperswap V2

`Hyperswap V2`, inspired by `Camelot`, provides a robust and efficient decentralized exchange platform. It utilizes dynamic liquidity pools to simplify token swaps and enhance user experience. This version focuses on accessibility and seamless integration with the `DeFi` ecosystem, making it a reliable solution for decentralized trading. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of Hyperswap V2

Item	Description
Name	Hyperswap
Type	Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	February 4, 2025

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- <https://github.com/HyperSwapX/contracts.git> (50bd223)

And here is the commit ID after all fixes for the issues found in the audit have been checked in:

- <https://github.com/HyperSwapX/contracts.git> (fc5b242)

## 1.2 About PeckShield

PeckShield Inc. [11] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (<https://t.me/peckshield>), Twitter (<http://twitter.com/peckshield>), or Email ([contact@peckshield.com](mailto:contact@peckshield.com)).

Table 1.2: Vulnerability Severity Classification

Impact				
	High	Medium	Low	
High	Critical	High	Medium	
Medium	High	Medium	Low	
Low	Medium	Low	Low	
		High	Medium	Low
		Likelihood		

## 1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact, and can be accordingly classified into four categories, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
Additional Recommendations	Holistic Risk Management
	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [9], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

## 1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit



Category	Summary
<b>Configuration</b>	Weaknesses in this category are typically introduced during the configuration of the software.
<b>Data Processing Issues</b>	Weaknesses in this category are typically found in functionality that processes data.
<b>Numeric Errors</b>	Weaknesses in this category are related to improper calculation or conversion of numbers.
<b>Security Features</b>	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
<b>Time and State</b>	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
<b>Error Conditions, Return Values, Status Codes</b>	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
<b>Resource Management</b>	Weaknesses in this category are related to improper management of system resources.
<b>Behavioral Issues</b>	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
<b>Business Logics</b>	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
<b>Initialization and Cleanup</b>	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
<b>Arguments and Parameters</b>	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
<b>Expression Issues</b>	Weaknesses in this category are related to incorrectly written expressions within code.
<b>Coding Practices</b>	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.



## 2 | Findings

### 2.1 Summary

Here is a summary of our findings after analyzing the implementation of the `Hyperswap V2` protocol. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	3	
Low	2	
Informational	0	
Total	5	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in [Section 3](#).

## 2.2 Key Findings

Overall, this smart contract is well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 3 medium-severity vulnerabilities and 2 low-severity vulnerabilities.

Table 2.1: Key Hyperswap V2 Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Improved <code>setStableSwap()</code> Logic in <code>HyperswapPair</code>	Business Logic	Confirmed
PVE-002	Low	Implicit Assumption Enforcement In <code>AddLiquidity()</code>	Business Logic	Resolved
PVE-003	Low	Improper LP Volume Accounting in <code>HyperswapRouter</code>	Business Logic	Resolved
PVE-004	Medium	Lack of Slippage Control in <code>MoneyMaker</code>	Time And State	Confirmed
PVE-005	Medium	Trust Issue of Admin Keys	Security Features	Mitigated

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contract is being deployed on mainnet. Please refer to Section 3 for details.

## 3 | Detailed Results

### 3.1 Improved setStableSwap() Logic in HyperswapPair

- ID: PVE-001
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: HyperswapPair
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

#### Description

The Hyperswap V2 protocol utilizes dynamic liquidity pools to simplify token swaps and enhance user experience. It has the support of specifying certain pairs to have tokens of the same type so that their swaps are followed with a more efficient stable swap curve. While reviewing the related logic, we notice current implementation can be improved especially for the accounting of LP fee.

To elaborate, we show below the related `setStableSwap()` routine. It has a rather straightforward logic in switching the given pool to use the efficient stable swap curve for a pair with similar tokens. However, our analysis shows that the `kLast` state should be computed when `feeOn` is true, not current `stable && feeOn` (line 117).

```

106  function setStableSwap(bool stable, uint112 expectedReserve0, uint112 expectedReserve1
    ) external lock {
107      require(msg.sender == IHyperswapFactory(factory).setStableOwner(), "HyperswapPair
        only factory's setStableOwner");
108      require(!pairTypeImmutable, "HyperswapPair immutable");
109
110      require(stable != stableSwap, "HyperswapPair no update");
111      require(expectedReserve0 == reserve0 && expectedReserve1 == reserve1, "HyperswapPair
        failed");
112
113      bool feeOn = _mintFee(reserve0, reserve1);
114
115      emit SetStableSwap(stableSwap, stable);
116      stableSwap = stable;
117      kLast = (stable && feeOn) ? _k(uint(reserve0), uint(reserve1)) : 0;

```

118 }

Listing 3.1: HyperswapPair::setStableSwap()

**Recommendation** Revisit the above routine to properly record the `kLast` state, which is important to accurately account for LP fees.

**Status** This issue has been confirmed.

## 3.2 Implicit Assumption Enforcement In AddLiquidity()

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: HyperswapRouter
- Category: Coding Practices [6]
- CWE subcategory: CWE-628 [2]

### Description

In the `HyperswapRouter` contract, the `addLiquidity()` routine (see the code snippet below) is provided to add `amountADesired` amount of `tokenA` and `amountBDesired` amount of `tokenB` into the pool as liquidity via the `HyperswapRouter::addLiquidity()` routine. To elaborate, we show below the related code snippet.

```

44  function _addLiquidity(
45      address tokenA,
46      address tokenB,
47      uint amountADesired,
48      uint amountBDesired,
49      uint amountAMin,
50      uint amountBMin
51  ) internal returns (uint amountA, uint amountB) {
52      // create the pair if it doesn't exist yet
53      if (IHyperswapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
54          IHyperswapFactory(factory).createPair(tokenA, tokenB);
55      }
56      (uint reserveA, uint reserveB) = UniswapV2Library.getReserves(factory, tokenA,
57          tokenB);
58      if (reserveA == 0 && reserveB == 0) {
59          (amountA, amountB) = (amountADesired, amountBDesired);
60      } else {
61          uint amountBOptimal = UniswapV2Library.quote(amountADesired, reserveA, reserveB);
62          if (amountBOptimal <= amountBDesired) {
63              require(amountBOptimal >= amountBMin, 'HyperswapRouter: INSUFFICIENT_B_AMOUNT');
64              (amountA, amountB) = (amountADesired, amountBOptimal);
65          } else {

```

```

65     uint amountAOptimal = UniswapV2Library.quote(amountBDesired, reserveB, reserveA)
66     ;
67     assert(amountAOptimal <= amountADesired);
68     require(amountAOptimal >= amountAMin, 'HyperswapRouter: INSUFFICIENT_A_AMOUNT');
69     (amountA, amountB) = (amountAOptimal, amountBDesired);
70 }
71 }

73 function addLiquidity(
74     address tokenA,
75     address tokenB,
76     uint amountADesired,
77     uint amountBDesired,
78     uint amountAMin,
79     uint amountBMin,
80     address to,
81     uint deadline
82 ) external override ensure(deadline) returns (uint amountA, uint amountB, uint
    liquidity) {
83     (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired,
        amountAMin, amountBMin);
84     address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
85     TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
86     TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
87     liquidity = IHyperswapPair(pair).mint(to);
88 }

```

Listing 3.2: HyperswapRouter::addLiquidity()

It comes to our attention that the HyperswapRouter has implicit assumptions on the `_addLiquidity()` routine. The above routine takes two amounts: `amountXDesired` and `amountXMin`. The first amount `amountXDesired` determines the desired amount for adding liquidity to the pool and the second amount `amountXMin` determines the minimum amount of used assets. There are two implicit conditions, i.e., `amountADesired >= amountAMin` and `amountBDesired >= amountBMin`. However, if these two conditions are not met, current logic will not trigger reverts because the code above performs asymmetric checks for these amounts. Hence, without stating these assumptions, slippage control for some trades on SquadSwap V2 Router may not be checked and may not be taken into account at all in certain scenarios.

**Recommendation** Make the requirement of `amountADesired >= amountAMin` and `amountBDesired >= amountBMin` explicitly in the `addLiquidity()` function.

**Status** This issue has been fixed in the following commit: [fc5b242](#).

### 3.3 Improper LP Volume Accounting in HyperswapRouter

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: HyperswapRouter
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

#### Description

The Hyperswap V2 protocol shares similar code layout from UniswapV2 with unique customization. While reviewing the customization on the HyperswapRouter contract, we notice the addition of `points` to keep track of trading/LP volume. Our analysis shows that related accounting logic may not be accurate and need to be revised.

To elaborate, we show below the related `addLiquidityETH()` routine. It has a rather straightforward logic in adding new liquidity to the specified pool. We notice the final update to the `points[msg.sender].lpVol` state (line 114) to record the new LP addition from the contributing user. However, if a user adds the same liquidity via `addLiquidity()` (for the same WETH-related pair), the user contribution is not recorded. Similarly, the `removeLiquidity()` routine is not revised to keep track of the liquidity removal either. Similarly, the swap-related volume accounting shares the same issue and is not incomplete.

```

90  function addLiquidityETH(
91      address token,
92      uint amountTokenDesired,
93      uint amountTokenMin,
94      uint amountETHMin,
95      address to,
96      uint deadline
97  ) external override payable ensure(deadline) returns (uint amountToken, uint amountETH
    , uint liquidity) {
98      (amountToken, amountETH) = _addLiquidity(
99          token,
100         WETH,
101         amountTokenDesired,
102         msg.value,
103         amountTokenMin,
104         amountETHMin
105     );
106     address pair = UniswapV2Library.pairFor(factory, token, WETH);
107     TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
108     IWETH(WETH).deposit{value : amountETH}();
109     assert(IWETH(WETH).transfer(pair, amountETH));
110     liquidity = IHyperswapPair(pair).mint(to);
111     // refund dust eth, if any

```

```

112     if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value -
        amountETH);
113
114     points[msg.sender].lpVol += msg.value;
115 }

```

Listing 3.3: HyperswapRouter::addLiquidityETH()

**Recommendation** Revisit the above-mentioned routines to properly record the swap/LP volume in HyperswapRouter.

**Status** This issue has been fixed in the following commit: [fc5b242](#).

### 3.4 Lack of Slippage Control in MoneyMaker

- ID: PVE-004
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: MoneyMaker
- Category: Time and State [8]
- CWE subcategory: CWE-682 [3]

#### Description

The Hyperswap V2 protocol has a key MoneyMaker contract that by design receives 0.05% of the swaps in the form of an LP. This contract allows to swap those LPs to a token of choice. While reviewing the token swap logic, we notice the current swap is obtained without enforcing necessary slippage control.

```

700     function _swap(
701         address fromToken,
702         address toToken,
703         uint256 amountIn,
704         address to,
705         uint256 slippage
706     ) internal returns (uint256 amountOut) {
707         // Checks
708         // X1 - X5: OK
709         IHyperswapPair pair = IHyperswapPair(factory.getPair(fromToken, toToken));
710         require(address(pair) != address(0), "MoneyMaker: Cannot convert");
711
712         (uint256 reserve0, uint256 reserve1, ,) = pair.getReserves();
713         (uint256 reserveInput, uint256 reserveOutput) = fromToken == pair.token0()
714             ? (reserve0, reserve1)
715             : (reserve1, reserve0);
716         IERC20(fromToken).safeTransfer(address(pair), amountIn);

```

```

717     uint256 amountInput = IERC20(fromToken).balanceOf(address(pair)).sub(
        reserveInput); // calculate amount that was transferred, this accounts for
        transfer taxes
718
719     if (fromToken != pair.token0()) (reserve0, reserve1) = (reserve1, reserve0);
720     amountOut = pair.getAmountOut(amountInput, fromToken);
721
722     (uint256 amount0Out, uint256 amount1Out) = fromToken == pair.token0()
723         ? (uint256(0), amountOut)
724         : (amountOut, uint256(0));
725     pair.swap(amount0Out, amount1Out, to, new bytes(0));
726 }

```

Listing 3.4: MoneyMaker::\_swap()

To elaborate, we show above the related swap routine. We notice the conversion is routed to UniswapV2 pair in order to determine the current swap price. Apparently, the instant DEX price is highly volatile and there is a need to consider the use of TWAP and further specify necessary restriction on the swap operation on possible slippage, so that it is not vulnerable to possible front-running attacks.

Note that this is a common issue plaguing current AMM-based DEX solutions. Specifically, a large trade may be sandwiched by a preceding sell to reduce the market price, and a tailgating buy-back of the same amount plus the trade amount. Such sandwiching behavior unfortunately causes a loss and brings a smaller return as expected to the trading user because the swap rate is lowered by the preceding sell. As a mitigation, we may consider specifying the restriction on possible slippage caused by the trade or referencing the TWAP or time-weighted average price of UniswapV2. Nevertheless, we need to acknowledge that this is largely inherent to current blockchain infrastructure and there is still a need to continue the search efforts for an effective defense.

**Recommendation** Develop an effective mitigation (e.g., slippage control) to the above front-running attack to better protect the interests of farming users.

**Status** This issue has been confirmed.

### 3.5 Trust Issue of Admin Keys

- ID: PVE-005
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: Multiple Contracts
- Category: Security Features [5]
- CWE subcategory: CWE-287 [1]



## Description

In the Hyperswap V2 protocol, there is a privileged account `owner` that plays a critical role in governing and regulating the system-wide operations (e.g., configure fee-related parameters and collect protocol fee). The account also has the privilege to control or govern the flow of assets managed by this protocol. Our analysis shows that the privileged account needs to be scrutinized. In the following, we examine the privileged account and the related privileged accesses in current contracts.

```

76     function setOwner(address _owner) external onlyOwner {
77         require(_owner != address(0), "HyperswapFactory zero address");
78         emit OwnershipTransferred(owner, _owner);
79         owner = _owner;
80     }
81
82     function setFeePercentOwner(address _feePercentOwner) external onlyOwner {
83         require(_feePercentOwner != address(0), "HyperswapFactory zero address");
84         emit FeePercentOwnershipTransferred(feePercentOwner, _feePercentOwner);
85         feePercentOwner = _feePercentOwner;
86     }
87
88     function setSetStableOwner(address _setStableOwner) external {
89         require(msg.sender == setStableOwner, "HyperswapFactory not setStableOwner");
90         require(_setStableOwner != address(0), "HyperswapFactory zero address");
91         emit SetStableOwnershipTransferred(setStableOwner, _setStableOwner);
92         setStableOwner = _setStableOwner;
93     }
94
95     function setFeeTo(address _feeTo) external onlyOwner {
96         emit FeeToTransferred(feeTo, _feeTo);
97         feeTo = _feeTo;
98     }

```

Listing 3.5: Example Privileged Functions in HyperswapFactory

Note that if these privileged accounts are plain EOA accounts, this may be worrisome and pose counter-party risk to the exchange users. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

**Recommendation** Promptly transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

**Status** This issue has been mitigated as the team makes use of a multisig to act as the privileged owner.

## 4 | Conclusion

In this audit, we have analyzed the design and implementation of the `Hyperswap V2` protocol, which is inspired by `Camelot` and aims to provide a robust and efficient decentralized exchange platform. It utilizes dynamic liquidity pools to simplify token swaps and enhance user experience. This version focuses on accessibility and seamless integration with the `DeFi` ecosystem, making it a reliable solution for decentralized trading. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that `Solidity`-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.



## References

- [1] MITRE. CWE-287: Improper Authentication. <https://cwe.mitre.org/data/definitions/287.html>.
- [2] MITRE. CWE-628: Function Call with Incorrectly Specified Arguments. <https://cwe.mitre.org/data/definitions/628.html>.
- [3] MITRE. CWE-682: Incorrect Calculation. <https://cwe.mitre.org/data/definitions/682.html>.
- [4] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. <https://cwe.mitre.org/data/definitions/841.html>.
- [5] MITRE. CWE CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [6] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [7] MITRE. CWE CATEGORY: Business Logic Errors. <https://cwe.mitre.org/data/definitions/840.html>.
- [8] MITRE. CWE CATEGORY: Error Conditions, Return Values, Status Codes. <https://cwe.mitre.org/data/definitions/389.html>.
- [9] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.

[10] OWASP. Risk Rating Methodology. [https://www.owasp.org/index.php/OWASP\\_Risk\\_Rating\\_Methodology](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology).

[11] PeckShield. PeckShield Inc. <https://www.peckshield.com>.

