SMART CONTRACT AUDIT REPORT

for

DCAv2

Prepared By: Yiqun Chen

PeckShield

December 18, 2021

Document Properties

Client	Mean Finance
Title	Smart Contract Audit Report
Target	DCAv2
Version	1.0
Author	Xuxian Jiang
Auditors	Stephen Bie, Yiqun Chen, Xuxian Jiang
Reviewed by	Yiqun Chen
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	December 18, 2021	Xuxian Jiang	Final Release

Contact

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

Name	Yiqun Chen
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the **DCAv2** design document and related smart contract source code, 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 Mean Finance

`Mean Finance` is the state-of-the-art protocol that is designed to enable users to `Dollar Cost Average (DCA)` any ERC20 into any ERC20 with their preferred period frequency by leveraging the `UniswapV3 TWAP` oracles. The purpose is to effectively neutralize the short-term volatility in the market. The purchases occur regardless of the asset's price and at regular intervals. In effect, this strategy removes much of the detailed work of attempting to time the market in order to make purchases of assets at the best prices. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of DCAv2

Item	Description
Name	Mean Finance
Website	https://mean.finance/
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	December 18, 2021

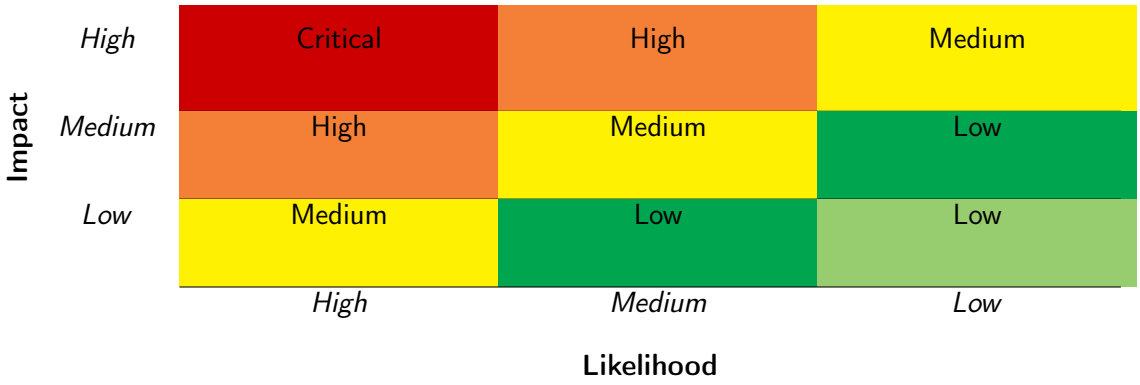
In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/Mean-Finance/dca-v2-core.git (e7e618e)

1.2 About PeckShield

PeckShield Inc. [9] 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).

Table 1.2: Vulnerability Severity Classification



1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [8]:

- 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 classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of 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 Audit Checklist

Category	Checklist Items
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
	Holistic Risk Management
Additional Recommendations	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) [7], 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
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	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.
Error Conditions, Return Values, Status Codes	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.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	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.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	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 DCAv2 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 logic, 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	1	■
Low	1	■
Informational	1	■
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. 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, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability, 2 low-severity vulnerabilities, and 1 informational recommendation.

Table 2.1: Key DCAv2 Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Improved Logic For Permission Accounting	Business Logic	Resolved
PVE-002	Informational	Consistency in reconfigureSupportForPair()	Coding Practices	Resolved
PVE-003	Medium	Trust Issue of Admin Keys	Security Features	Resolved

Besides 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 contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Improved Logic For Permission Accounting

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Target: DCAPermissionsManager
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

The DCAv2 protocol has a key DCAPermissionsManager contract that is designed to manage various token-related permissions. While examining its logic, we notice it maintains the timestamp when a specific permission is assigned. Combined with the timestamp upon the creation or ownership transfer of an ERC721 token, the protocol can precisely revoke the stale permissions that were assigned by the previous owner(s).

To elaborate, we show below the related _beforeTokenTransfer() routine, a hook function that will be called before any token transfer. Considering the possibility of re-minting a burnt tokenID or transfering to the owner herself, it is helpful to improve the routine by replacing the if (_from != address(0)) (line 198) with if (_from != to).

```
190   function _beforeTokenTransfer(
191     address _from,
192     address _to,
193     uint256 _id
194   ) internal override {
195     if (_to == address(0)) {
196       // When token is being burned, we can delete this entry on the mapping
197       delete lastOwnershipChange[_id];
198     } else if (_from != address(0)) {
199       // If the token is being minted, then no need to write this
200       lastOwnershipChange[_id] = uint32(block.timestamp);
201     }
```

```
202   }
```

Listing 3.1: DCAPermissionsManager::_beforeTokenTransfer()

Recommendation Properly keep track of the lastOwnershipChange timestamp for each minted tokenID.

Status This issue has been resolved as the team explains that the same tokenID cannot be re-minted. And it makes sense to avoid writing to storage and reduce the gas cost of updating the lastOwnershipChange state.

3.2 Consistency in reconfigureSupportForPair()

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: Multiple Contracts
- Category: Coding Practices [5]
- CWE subcategory: CWE-1126 [1]

Description

The DCAv2 protocol has the built-in oracle support of both Chainlink and UniswapV3. To seamlessly integrate their support, the protocol also develops an aggregate contract to combine their benefits. While examining the related oracle contracts, we notice an inconsistency in the exposed functions.

To elaborate, we show below the reconfigureSupportForPair() function in these three contracts: ChainlinkOracle, UniswapV3Oracle, and OracleAggregator. It comes to our attention that this function is perimssionless in the first two contracts while it becomes perimssioned in the third contract. This inconsistency may be confusing and it is helpful to resolve the inconsistency.

```
78    /// @inheritdoc IPriceOracle
79    function reconfigureSupportForPair(address _tokenA, address _tokenB) external {
80      _addSupportForPair(_tokenA, _tokenB);
81    }
82
83    /// @inheritdoc IPriceOracle
84    function addSupportForPairIfNeeded(address _tokenA, address _tokenB) external {
85      (address __tokenA, address __tokenB) = TokenSorting.sortTokens(_tokenA, _tokenB);
86      if (planForPair[__tokenA][__tokenB] == PricingPlan.NONE) {
87        _addSupportForPair(_tokenA, _tokenB);
88      }
89    }
```

Listing 3.2: ChainlinkOracle::reconfigureSupportForPair()

```
57  /// @inheritdoc IPriceOracle
58  function reconfigureSupportForPair(address _tokenA, address _tokenB) external override
        {
59    (address __tokenA, address __tokenB) = TokenSorting.sortTokens(_tokenA, _tokenB);
60    delete _poolsForPair[__tokenA][__tokenB];
61    _addSupportForPair(__tokenA, __tokenB);
62  }
63
64  /// @inheritdoc IPriceOracle
65  function addSupportForPairIfNeeded(address _tokenA, address _tokenB) external override
        {
66    (address __tokenA, address __tokenB) = TokenSorting.sortTokens(_tokenA, _tokenB);
67    if (_poolsForPair[__tokenA][__tokenB].length == 0) {
68      _addSupportForPair(__tokenA, __tokenB);
69    }
70  }
```

Listing 3.3: `UniswapV3Oracle::reconfigureSupportForPair()`

```
53  /// @inheritdoc IPriceOracle
54  function reconfigureSupportForPair(address _tokenA, address _tokenB) external
        onlyGovernor {
55    (address __tokenA, address __tokenB) = TokenSorting.sortTokens(_tokenA, _tokenB);
56    _addSupportForPair(__tokenA, __tokenB);
57  }
58
59  /// @inheritdoc IPriceOracle
60  function addSupportForPairIfNeeded(address _tokenA, address _tokenB) external {
61    (address __tokenA, address __tokenB) = TokenSorting.sortTokens(_tokenA, _tokenB);
62    if (_oracleInUse[__tokenA][__tokenB] == OracleInUse.NONE) {
63      _addSupportForPair(__tokenA, __tokenB);
64    }
65  }
```

Listing 3.4: `OracleAggregator::reconfigureSupportForPair()`

Recommendation Maintain an consistent interface for the exported functions in current oracle contracts.

Status The issue has been confirmed and the team clarifies the intention to allow the oracle aggregator to choose between the given `oracle1` and `oracle2`. In other words, the permissioned design in `OracleAggregator` is intentional.

3.3 Trust Issue of Admin Keys

- ID: PVE-003
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: Multiple Contracts
- Category: Security Features [4]
- CWE subcategory: CWE-287 [2]

Description

In the DCAv2 protocol, there is a privileged admin account (with the role of TIME_LOCKED_ROLE) that plays a critical role in governing and regulating the protocol-wide operations (e.g., configuring various system parameters). It 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 their related privileged accesses in current contracts.

```
56  /// @inheritdoc IDCAHubConfigHandler
57  function setSwapFee(uint32 _swapFee) external onlyRole(TIME_LOCKED_ROLE) {
58    _validateFee(_swapFee);
59    swapFee = _swapFee;
60    emit SwapFeeSet(_swapFee);
61  }
62
63  /// @inheritdoc IDCAHubConfigHandler
64  function setLoanFee(uint32 _loanFee) external onlyRole(TIME_LOCKED_ROLE) {
65    _validateFee(_loanFee);
66    loanFee = _loanFee;
67    emit LoanFeeSet(_loanFee);
68  }
69
70  /// @inheritdoc IDCAHubConfigHandler
71  function setPlatformFeeRatio(uint16 _platformFeeRatio) external onlyRole(
        TIME_LOCKED_ROLE) {
72    if (_platformFeeRatio > MAX_PLATFORM_FEE_RATIO) revert HighPlatformFeeRatio();
73    platformFeeRatio = _platformFeeRatio;
74    emit PlatformFeeRatioSet(_platformFeeRatio);
75  }
```

Listing 3.5: Example Setters in the DCAHubConfigHandler Contract

Apparently, the admin account that is able to adjust various protocol-wide risk parameters. If the privileged admin account is a plain EOA account, this may be worrisome and pose counter-party risk to the protocol users. Note that 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.

Moreover, it should be noted that if current contracts need to be deployed behind a proxy, there is a need to properly manage the proxy-admin privileges as they fall in this trust issue as well.

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 confirmed and partially mitigated. Especially, for all admin-level operations, the current mitigation is to adopt the standard `Timelock` contract, which is enforced as part of the deployment script.

4 Conclusion

In this audit, we have analyzed the design and implementation of the DCAv2 protocol, which is the state-of-the-art protocol that is designed to enable users to Dollar Cost Average (DCA) any ERC20 into any ERC20 with their preferred period frequency by leveraging the UniswapV3 TWAP oracles. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and fixed.

Moreover, 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-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

[4] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[5] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[6] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[7] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[8] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[9] PeckShield. PeckShield Inc. https://www.peckshield.com.