SMART CONTRACT AUDIT REPORT

for

Tradao Order Module

Prepared By: Xiaomi Huang

PeckShield

December 14, 2023

Document Properties

Client	Tradao
Title	Smart Contract Audit Report
Target	Tradao
Version	1.0
Author	Xuxian Jiang
Auditors	Colin Zhong, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	December 14, 2023	Xuxian Jiang	Final Release
1.0-rc1	December 12, 2023	Xuxian Jiang	Release Candidate #1

Contact

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

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the Tradao 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 audited protocol 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 Tradao

Tradao is a Web3 Onchain Derivatives Portfolio Tracker that empowers traders with a comprehensive toolset and an innovative incentive system, such as copy trading, backtesting, grid trading bot, etc. The platform is designed to enable traders to track and capitalize on related data from GMX, Kwenta , and other derivatives protocols across various blockchain networks, including Arbitrum, Optimism, BNB Chain, Avalanche, Ton and more. By harnessing the potential of decentralized technology, the goal here is to establish a fair and transparent trading environment, providing real-time signals that facilitate informed investment decisions for traders of all levels. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of Tradao

Item	Description
Name	Tradao
Type	EVM Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	December 14, 2023

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

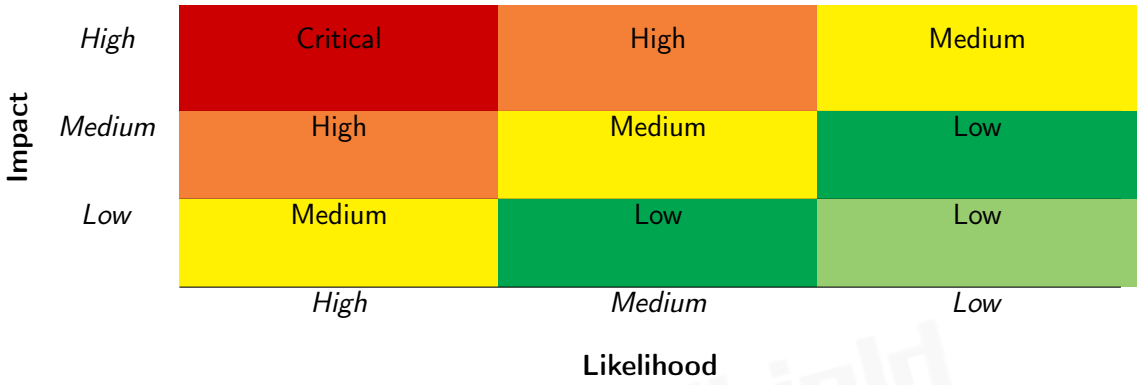
this audit. This audit only covers two following contracts: `BiconomyModuleSetup` and `Gmxv2OrderModule`.

- https://github.com/tradao-xyz/TradaoOrderModule.git (cea8d8e)

1.2 About PeckShield

PeckShield Inc. [7] 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 OWASP Risk Rating Methodology [6]:

- 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 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

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
	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

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 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) [5], 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.

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 Logics	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 Tradao implementation. 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	0	
Informational	1	■
Total	2	

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 that 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 and 1 informational recommendation.

Table 2.1: Key Tradao Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Suggested OperatorTransferred Event in Constructor	Coding Practices	Confirmed
PVE-002	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 contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Suggested OperatorTransferred Event in Constructor

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `SynthChefV2`
- Category: Coding Practices [4]
- CWE subcategory: CWE-1126 [1]

Description

In `Ethereum`, the `event` is an indispensable part of a contract and is mainly used to record a variety of runtime dynamics. In particular, when an `event` is emitted, it stores the arguments passed in transaction logs and these logs are made accessible to external analytics and reporting tools. `Events` can be emitted in a number of scenarios. One particular case is when system-wide parameters or settings are being changed. Another case is when tokens are being minted, transferred, or burned.

In the following, we use the `Gmxv2OrderModule` contract as an example. This contract has public functions that are used to updated various risk parameters. While examining the event that reflects the `operator` change, we notice the constructor is suggested to emit related event `emit OperatorTransferred(address(0), initialOperator)`, which is currently missing.

```
106     constructor(address initialOperator) Ownable(msg.sender) {
107         operator = initialOperator;
108     }
110     function transferOperator(address newOperator) external onlyOwner {
111         address oldOperator = operator;
112         operator = newOperator;
113         emit OperatorTransferred(oldOperator, newOperator);
114     }
```

Listing 3.1: `Gmxv2OrderModule::updateEmissionRateCommunity()`

Recommendation Improve the constructor to emit the respective `OperatorTransferred` event when a new `operator` becomes effective.

Status This issue has been confirmed.

3.2 Trust Issue of Admin Keys

- ID: PVE-002
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `Multiple Contracts`
- Category: Security Features [3]
- CWE subcategory: CWE-287 [2]

Description

In the `Tradao` protocol, there is a privileged account (`owner`). This account plays critical roles in governing and regulating the protocol-wide operations (e.g., configure protocol parameters and assign the operator role). Our analysis shows that the privileged account needs to be scrutinized. In the following, we use the `Gmxv2OrderModule` contract as an example and show the representative functions potentially affected by the privileged account.

```
110     function transferOperator(address newOperator) external onlyOwner {
111         address oldOperator = operator;
112         operator = newOperator;
113         emit OperatorTransferred(oldOperator, newOperator);
114     }

116     function setReferralCode(address smartAccount) public onlyOperator returns (bool
            isSuccess) {
117         return IModuleManager(smartAccount).execTransactionFromModule(
118             REFERRALSTORAGE, 0, SETREFERRALCODECALLDATA, Enum.Operation.Call
119         );
120     }

122     function updateTxGasFactor(uint256 _txGasFactor) external onlyOperator {
123         require(_txGasFactor <= MAX_TXGAS_FACTOR, "400");
124         uint256 _prevFactor = txGasFactor;
125         txGasFactor = _txGasFactor;
126         emit TxGasFactorUpdated(_prevFactor, _txGasFactor);
127     }

129     function cancelOrder(address smartAccount, bytes32 key) external onlyOperator
            returns (bool success) {...}

131     function newOrder(uint256 triggerPrice, OrderParamBase memory _orderBase, OrderParam
             memory _orderParam)
132         external
```

```
133         onlyOperator
134         returns (bytes32 orderKey)
135     {...}

137     function newOrders(OrderParamBase memory _orderBase, OrderParam[] memory orderParams
            )
138         external
139         onlyOperator
140         returns (bytes32[] memory orderKeys)
141     {...}
```

Listing 3.2: Example Privileged Operations in `Gmxv2OrderModule`

We understand the need of the privileged functions for proper contract operations, but at the same time the extra power to the privileged accounts may also be a counter-party risk to the contract users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Recommendation Promptly transfer the administrative privileges to the intended DAO-like governance contract. And activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status The issue has been mitigated as the team confirmed that all the privileged accounts will be multi-sig wallets.

4 Conclusion

In this audit, we have analyzed the design and implementation of the Tradao protocol, which is a Web3 Onchain Derivatives Portfolio Tracker that empowers traders with a comprehensive toolset and an innovative incentive system, such as copy trading, backtesting, grid trading bot, etc. The platform is designed to enable traders to track and capitalize on related data from GMX, Kwenta, and other derivatives protocols across various blockchain networks, including Arbitrum, Optimism, BNB Chain, Avalanche, Ton and more. By harnessing the potential of decentralized technology, the goal here is to establish a fair and transparent trading environment, providing real-time signals that facilitate informed investment decisions for traders of all levels. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that 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 CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

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

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

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

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