SMART CONTRACT AUDIT REPORT

for

SeiNativeOracleAdapter

Prepared By: Xiaomi Huang

PeckShield

November 22, 2024

Document Properties

Client	DragonSwap
Title	Smart Contract Audit Report
Target	SeiNativeOracleAdapter
Version	1.0-rc
Author	Xuxian Jiang
Auditors	Daisy Cao, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Confidential

Version Info

Version	Date	Author(s)	Description
1.0-rc	November 21, 2024	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 `Sui Native Oracle Adapter` in `DragonSwap`, 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 current implementation. This document outlines our audit results.

1.1 About DragonSwap

`DragonSwap` is the first native `DEX` on `Sei` network, marking a significant evolution with its launch. The audited contract is a library that enables an out-of-the-box utilization of the `Sei Native Oracle` though the `Solidity` smart-contracts. The `Sei Native Oracle` returns `18` decimal fixed point numbers in a string format (which is usually not directly usable), and this library helps you easily retrieve exchange rates in `uint256` format with a manageable amount decimals. The basic information of the audited contract is as follows:

Table 1.1: Basic Information of SeiNativeOracleAdapter

Item	Description
Protocol Name	DragonSwap
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	November 22, 2024

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

- https://github.com/dragonswap-app/sei-native-oracle-adapter.git (19b525d)

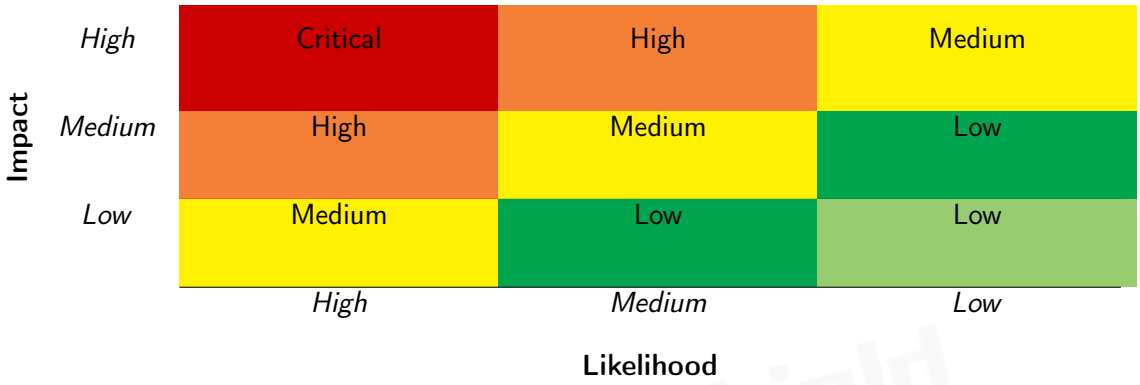
And here is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/dragonswap-app/sei-native-oracle-adapter.git (TBD)

1.2 About PeckShield

PeckShield Inc. [6] 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 [5]:

- 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

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

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) [4], 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.

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.

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.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the implementation of the `Sui Native Oracle Adapter` in `DragonSwap`. 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	0	
Low	1	■
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 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 low-severity vulnerability and 1 informational recommendation.

Table 2.1: Key SeiNativeOracleAdapter Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Early Exit in Rate/TWAP Calculation Logic	Coding Practices	
PVE-002	Informational	Redundant Code/State Removal	Coding Practices	

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 Early Exit in Rate/TWAP Calculation Logic

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `SeiNativeOracleAdapter`
- Category: Coding Practices [3]
- CWE subcategory: CWE-1126 [1]

Description

As mentioned earlier, the audited `SeiNativeOracleAdapter` contract is designed to wrap the `Sei Native Oracle` to retrieve exchange rates in `uint256` format with a manageable amount decimals instead of the original string format. While reviewing the related logic, we notice current implementation can be improved.

To elaborate, we show below the implementation of the related `setReserveInterestRateStrategyAddress()` routine. As the name indicates, it is used to retrieve the time weighed average price for the given token and time period. It comes to our attention that current implementation iterates the full returned price list from the native oracle. An optimization can be placed to make an early exit once there is a match on the `denom` string hashes. The same optimization can be applied to another similar function named `getExchangeRate()`.

```
64      function getOracleTwap(string memory denom, uint64 lookbackSeconds) internal view
            returns (uint256 twap) {
65          // Retrieve twap values in the default/string format from the native oracle.
66          ISeiNativeOracle.OracleTwap[] memory data = NATIVE_ORACLE.getOracleTwaps(
                lookbackSeconds);
67          // Gas opt.
68          uint256 length = data.length;
69          for (uint256 i; i < length; ++i) {
70              // Compare string hashes and proceed once the matching occurs.
71              if (keccak256(bytes(data[i].denom)) == keccak256(bytes(denom))) {
72                  // Return converted twap value.
73                  twap = convertStringNumberToUint256(data[i].twap);
```

```
74              }
75          }
76      }
```

Listing 3.1: SeiNativeOracleAdapter::getOracleTwap()

Recommendation Revise the above-mentioned routines to make an early exit optimization once the intended rate or TWAP is retrieved.

Status

3.2 Redundant Code/State Removal

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: SeiNativeOracleAdapter
- Category: Coding Practices [3]
- CWE subcategory: CWE-563 [2]

Description

To facilitate the implementation, the SeiNativeOracleAdapter contract has a number of helper routines. While reviewing these helper routines, we notice a specific one (that is used to to change the decimals), which is currently not used and can be safely removed.

In particular, we show below the implementation of this specific function, i.e., changeDecimals(). It has a rather straightforward logic in trimming or extending the decimals for the conversion of retrieved exchange rates. Since it is no longer used, we can safely remove it from current contract.

```
164     function changeDecimals(uint256 number, uint256 fromDecimals, uint256 toDecimals)
            internal pure returns (uint256) {
165         // Compare decimals.
166         if (toDecimals > fromDecimals) {
167             // Append zeros.
168             number *= 10 ** (toDecimals - fromDecimals);
169         } else if (fromDecimals > toDecimals) {
170             // Trim decimals.
171             number /= 10 ** (fromDecimals - toDecimals);
172         }
173         return number;
174     }
```

Listing 3.2: SeiNativeOracleAdapter::changeDecimals()

Recommendation Consider the removal of the redundant state (or code) with a simplified, consistent implementation.

Status

4 Conclusion

In this audit, we have analyzed the design and implementation of the Sui Native Oracle Adapter in DragonSwap, which is the first native DEX on Sei network, marking a significant evolution with its launch. The audited contract is a library that enables an out-of-the-box utilization of the Sei Native Oracle though the Solidity smart-contracts. The Sei Native Oracle returns 18 decimal fixed point numbers in a string format (which is usually not directly usable), and this library helps you easily retrieve exchange rates in uint256 format with a manageable amount decimals. 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-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

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

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

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

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