SMART CONTRACT AUDIT REPORT

for

Spot Vaults

Prepared By: Xiaomi Huang

PeckShield

June 16, 2024

Document Properties

Client	Fragments, Inc.
Title	Smart Contract Audit Report
Target	Spot Vaults
Version	1.0
Author	Xuxian Jiang
Auditors	Jason Shen, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	June 16, 2024	Xuxian Jiang	Final Release
1.0-rc1	June 10, 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 Spot Vaults 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 Spot Vaults

Spot is a decentralized, inflation resistant store of value designed to be resilient in all market conditions. It has no reliance on centralized custodians, liquidations, or lenders of last resort. Spot can be held directly or rotated in as an alternative collateral asset to USDC within existing systems. It uses AMPL as the underlying unit of account, Buttonwood Tranche for collateral preparation, and onchain governance through the FORTH DAO. This audit covers the Spot Vaults protocol that uses perpetual note tokens and dollars as available liquidity to facilitate the swaps. The swap may be charged with a fee (credited to the vault LPs), which is a function of available liquidity held in the protocol. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of The Spot Vaults Protocol

Item	Description
Client	Fragments, Inc.
Website	https://ampleforth.org
Type	EVM Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	June 16, 2024

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

- https://github.com/ampleforth/spot.git (17622f2)

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

- https://github.com/ampleforth/spot.git (3066098)

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

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

1.3 Methodology

To standardize the evaluation, we define the following terminology based on 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 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 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.

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 1.3: 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

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 design and implementation of the `Spot Vaults` 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	0	
Low	2	■ ■
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 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 2 low-severity vulnerabilities and and 1 informational recommendation.

Table 2.1: Key Spot Vaults Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Possible Costly BillBroker LP From Improper Initialization	Time and State	Resolved
PVE-002	Informational	Improved Validations of Protocol Parameters in BillBroker	Coding Practices	Resolved
PVE-003	Low	Trust on Admin Keys	Security Features	Resolved

Besides recommending specific countermeasures to mitigate these issues, we also emphasize that 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 need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Possible Costly BillBroker LP From Improper Initialization

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `BillBroker`
- Category: Time and State [5]
- CWE subcategory: CWE-362 [3]

Description

The `Spot Vaults` protocol allows users to deposit supported perp/USD tokens and get in return the vault LP tokens to represent the overall share/ownership. While examining the related deposit logic, we notice an issue that may unnecessarily make the vault LP token extremely expensive and bring hurdles (or even causes loss) for later depositors.

To elaborate, we show below the `deposit()` routine, which is used for participating users to deposit the perp/USD tokens and get respective vault LP tokens in return. The issue occurs when the `BillBroker` is being initialized under the assumption that it is empty.

```
    function deposit(
        uint256 usdAmtMax,
        uint256 perpAmtMax,
        uint256 usdAmtMin,
        uint256 perpAmtMin
    ) external nonReentrant whenNotPaused returns (uint256 mintAmt) {
        uint256 usdAmtIn;
        uint256 perpAmtIn;
        (mintAmt, usdAmtIn, perpAmtIn) = computeMintAmt(usdAmtMax, perpAmtMax);
        if (mintAmt <= 0) {
            return 0;
        }
        if (usdAmtIn < usdAmtMin  perpAmtIn < perpAmtMin) {
            revert SlippageTooHigh();
        }
```

```
        // Transfer perp and usd tokens from the user
        usd.safeTransferFrom(_msgSender(), address(this), usdAmtIn);
        perp.safeTransferFrom(_msgSender(), address(this), perpAmtIn);

        // mint LP tokens
        _mint(_msgSender(), mintAmt);
    }
```

Listing 3.1: `BillBroker::deposit()`

```
449     function computeMintAmt(
450         uint256 usdAmtMax,
451         uint256 perpAmtMax
452     ) public view returns (uint256 mintAmt, uint256 usdAmtIn, uint256 perpAmtIn) {
453         if (usdAmtMax <= 0 && perpAmtMax <= 0) {
454             return (0, 0, 0);
455         }

457         uint256 totalSupply_ = totalSupply();
458         // During the initial deposit we deposit the entire available amounts.
459         // The onus is on the depositor to ensure that the value of USD tokens and
460         // perp tokens on first deposit are equivalent.
461         if (totalSupply_ <= 0) {
462             usdAmtIn = usdAmtMax;
463             perpAmtIn = perpAmtMax;
464             mintAmt = (ONE.mulDiv(usdAmtIn, usdUnitAmt) +
465                 ONE.mulDiv(perpAmtIn, perpUnitAmt));
466             mintAmt = mintAmt * INITIAL_RATE;
467         }
468         ...
469     }
```

Listing 3.2: `BillBroker::computeMintAmt()`

Specifically, when it is being initialized (line 461), the minted amount is based on the given `usdAmtIn/perpAmtIn`, which could be manipulatable by the malicious actor. As this is the first deposit, the current total supply equals 0 and the return amount is computed as `(ONE.mulDiv(usdAmtIn, usdUnitAmt)+ ONE.mulDiv(perpAmtIn, perpUnitAmt))* INITIAL_RATE`. With that, the actor can further donate a huge amount of the underlying assets with the goal of making the perps extremely expensive.

An extremely expensive vault tokens can be very inconvenient to use as a small number of 1 `Wei` may denote a large value. Furthermore, it can lead to precision issue in truncating the vault tokens for deposited assets. If truncated to be zero, the deposited assets are essentially considered dust and kept by the pool without returning any pool tokens. Fortunately, current implementation supports necessary slippage control and disallows any zero-share amount.

This is a known issue that has been mitigated in popular `Uniswap`. When providing the initial liquidity to the contract (i.e. when totalSupply is 0), the liquidity provider must sacrifice 1000 LP tokens (by sending them to $address(0)$). By doing so, we can ensure the granularity of the LP tokens is always at least 1000 and the malicious actor is not the sole holder. This approach may bring an additional cost for the initial liquidity provider, but this cost is expected to be low and acceptable.

Recommendation Revise current deposit logic to defensively calculate the vault token amount when it is being initialized. An alternative solution is to ensure a guarded launch process that safeguards the first deposit to avoid being manipulated.

Status This issue has been fixed by this commit: `50ca6da`.

3.2 Improved Validations of Protocol Parameters in BillBroker

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `BillBroker`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

DeFi protocols typically have a number of system-wide parameters that can be dynamically configured on demand. The `Spot Vaults` protocol is no exception. Specifically, if we examine the `BillBroker` contract, it has defined a number of protocol-wide risk parameters, such as `mintFeePerc` and `burnFeePerc`. In the following, we show the corresponding routines that allow for their changes.

```
    function updateFees(BillBrokerFees memory fees_) public onlyOwner {
        if (
            fees_.mintFeePerc > ONE
            fees_.burnFeePerc > ONE
            fees_.perpToUSDSwapFeePercs.lower > fees_.perpToUSDSwapFeePercs.upper
            fees_.usdToPerpSwapFeePercs.lower > fees_.usdToPerpSwapFeePercs.upper
            fees_.protocolSwapSharePerc > ONE
        ) {
            revert InvalidPerc();
        }

        fees = fees_;
    }
```

Listing 3.3: `BillBroker::updateFees()`

These parameters define various aspects of the protocol operation and maintenance and need to exercise extra care when configuring or updating them. Our analysis shows the update logic on these parameters can be improved by applying more rigorous sanity checks. Based on the current implementation, certain corner cases may lead to an undesirable consequence. For example, the above `updateFees()` setter can be strengthened to ensure both `fees_.perpToUSDSwapFeePercs.upper` and `fees_.usdToPerpSwapFeePercs.upper` will be no larger than `ONE`.

Recommendation Validate any changes regarding these system-wide parameters to ensure they fall in an appropriate range. If necessary, also consider emitting relevant events for their changes.

Status This issue has been fixed by this commit: `dd68fb6`.

3.3 Trust Issue of Admin Keys

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

Description

In the `Spot Vaults` protocol, there is a special administrative account, i.e., `owner`. This `owner` account plays a critical role in governing and regulating the system-wide operations (e.g., configure various settings and execute privileged operations). It also has the privilege to control or govern the flow of assets within the protocol contracts. In the following, we examine the privileged account and the related privileged accesses in current contracts.

```
    function updateKeeper(address keeper_) public onlyOwner {
        keeper = keeper_;
    }

    /// @notice Updates the reference to the pricing strategy.
    /// @param pricingStrategy_ The address of the new pricing strategy.
    function updatePricingStrategy(
        IBillBrokerPricingStrategy pricingStrategy_
    ) public onlyOwner {
        if (pricingStrategy_.decimals() != DECIMALS) {
            revert UnexpectedDecimals();
        }
        pricingStrategy = pricingStrategy_;
    }

    /// @notice Updates the system fees.
    /// @param fees_ The new system fees.
    function updateFees(BillBrokerFees memory fees_) public onlyOwner {
        if (
            fees_.mintFeePerc > ONE
            fees_.burnFeePerc > ONE
            fees_.perpToUSDSwapFeePercs.lower > fees_.perpToUSDSwapFeePercs.upper
```

```
            fees_.usdToPerpSwapFeePercs.lower > fees_.usdToPerpSwapFeePercs.upper
            fees_.protocolSwapSharePerc > ONE
        ) {
            revert InvalidPerc();
        }

        fees = fees_;
    }

    /// @notice Updates the hard asset ratio bound.
    /// @dev Swaps are made expensive when the system is outside the defined soft bounds
        ,
    ///      and swaps are disabled when the system is outside the defined hard bounds.
    /// @param arSoftBound_ The updated soft bounds.
    /// @param arHardBound_ The updated hard bounds.
    function updateARBounds(
        Range memory arSoftBound_,
        Range memory arHardBound_
    ) public onlyOwner {
        bool validBounds = (arHardBound_.lower <= arSoftBound_.lower &&
            arSoftBound_.lower <= arSoftBound_.upper &&
            arSoftBound_.upper <= arHardBound_.upper);
        if (!validBounds) {
            revert InvalidARBound();
        }
        arSoftBound = arSoftBound_;
        arHardBound = arHardBound_;
    }

    //-----------------------------------------------------------------------------
    // Keeper only methods

    /// @notice Pauses deposits, withdrawals and swaps.
    /// @dev ERC-20 functions, like transfers will always remain operational.
    function pause() external onlyKeeper {
        _pause();
    }

    /// @notice Unpauses deposits, withdrawals and rollovers.
    /// @dev ERC-20 functions, like transfers will always remain operational.
    function unpause() external onlyKeeper {
        _unpause();
    }
```

Listing 3.4: Example Privileged Operations in `BillBroker`

We understand the need of the privileged functions for proper contract operations, but at the same time the extra power to these 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 privileged account to the intended DAO-like governance contract. All changes 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 The issue has been mitigated as the team clarifies that the `owner` account is currently a 2/4 DAO multisig. The ownership will eventually be handed off to `ForthDAO` governance + timelock the same as the `AMPL` contracts.

4 Conclusion

In this audit, we have analyzed the design and implementation of the `Spot Vaults` protocol. Note `Spot` is a decentralized, inflation resistant store of value designed to be resilient in all market conditions. It has no reliance on centralized custodians, liquidations, or lenders of last resort. `Spot` can be held directly or rotated in as an alternative collateral asset to `USDC` within existing systems. It uses `AMPL` as the underlying unit of account, `Buttonwood Tranche` for collateral preparation, and onchain governance through the `FORTH DAO`. This audit covers the `Spot Vaults` protocol that uses perpetual note tokens and dollars as available liquidity to facilitate the swaps. The swap may be charged with a fee (credited to the vault LPs), which is a function of available liquidity held in the protocol. The current code base is clearly organized and those identified issues are promptly confirmed and fixed.

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-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-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition'). https://cwe.mitre.org/data/definitions/362.html.

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

[5] MITRE. CWE CATEGORY: 7PK - Time and State. https://cwe.mitre.org/data/definitions/361.html.

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