



SMART CONTRACT AUDIT REPORT

for

Deri Protocol V4



Prepared By: Xiaomi Huang

PeckShield
October 9, 2023

Document Properties

| | |
|----------------|---|
| Client | Deri Protocol V4 |
| Title | Smart Contract Audit Report |
| Target | Deri-V4 |
| Version | 1.0 |
| Author | Xuxian Jiang |
| Auditors | Jinzhou Shen, Colin Zhong, Xuxian Jiang |
| Reviewed by | Patrick Lou |
| Approved by | Xuxian Jiang |
| Classification | Public |

Version Info

| Version | Date | Author(s) | Description |
|---------|-----------------|--------------|----------------------|
| 1.0 | October 9, 2023 | Xuxian Jiang | Final Release |
| 1.0-rc1 | October 5, 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 | 4 |
| 1.1 | About Deri-V4 | 4 |
| 1.2 | About PeckShield | 5 |
| 1.3 | Methodology | 5 |
| 1.4 | Disclaimer | 7 |
| 2 | Findings | 9 |
| 2.1 | Summary | 9 |
| 2.2 | Key Findings | 10 |
| 3 | Detailed Results | 11 |
| 3.1 | Improved vBNB Initialization in VaultImplementationVenus | 11 |
| 3.2 | Possible Precision Issue in DToken Redemption | 12 |
| 3.3 | Possible Costly DToken From Improper Vault Initialization | 13 |
| 3.4 | Trust Issue of Admin Keys | 15 |
| 4 | Conclusion | 17 |
| | References | 18 |

1 | Introduction

Given the opportunity to review the design document and related smart contract source code of the `Deri-V4` 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 Deri-V4

`Deri` is a decentralized protocol for users to exchange risk exposures precisely and capital-efficiently. The audited `Deri-V4` protocol is upgraded from `v3` with the adoption of the `Cross-Chain Decentralized Application (xDapp)` model. By constructing an all-chain decentralized protocol for derivative trading, it greatly enhances inclusivity, capital efficiency, and user experience, overcomes previous limitations, and sets a new standard for decentralized derivatives trading. The basic information of the `Deri` protocol is as follows:

Table 1.1: Basic Information of The `Deri` Protocol

| Item | Description |
|---------------------|---|
| Name | Deri Protocol V4 |
| Website | https://deri.io |
| Type | Ethereum Smart Contract |
| Platform | Solidity |
| Audit Method | Whitebox |
| Latest Audit Report | October 9, 2023 |

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit. Note that `Deri-V4` assumes a trusted price oracle with timely market price feeds for

supported assets and the oracle itself is not part of this audit.

- <https://github.com/deri-protocol/deriprotocol-v4.git> (65fae6b)

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

- <https://github.com/deri-protocol/deriprotocol-v4.git> (a23ba59)

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).

Table 1.2: Vulnerability Severity Classification

| | | | | |
|--------|--------|------------|--------|--------|
| Impact | 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.

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 |

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) [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.

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 `Deri-v4` 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 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 | 1 |  |
| Low | 3 |  |
| Informational | 0 | |
| Total | 4 | |

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 3 low-severity vulnerabilities.

Table 2.1: Key Deri-V4 Audit Findings

| ID | Severity | Title | Category | Status |
|---------|----------|---|-------------------|-----------|
| PVE-001 | Low | Improved vBNB Initialization in VaultImplementationVenus | Business Logic | Resolved |
| PVE-002 | Low | Possible Precision Issue in DToken Redemption | Numeric Errors | Resolved |
| PVE-003 | Low | Possible Costly DToken From Improper Vault Initialization | Coding Practices | Resolved |
| PVE-004 | Medium | Trust Issue of Admin Keys | Security Features | Mitigated |

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 Improved vBNB Initialization in VaultImplementationVenus

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

Description

To improve capital efficiency, Deri-V4 automatically deposits the funds into Venus for interest accrual. In the process of examining supported Venus markets, we notice the support of vBNB market can be improved.

To elaborate, we show below the implementation of the related constructor routine. This routine validates the given Venus market and configures the associated gateway, comptroller, and rewardToken addresses. However, we notice the vBNB market does not have the underlying field, which suggests the need of explicitly validating the given market argument to be vBNB: 0xa07c5b74c9b40447a954e1466938b865b6bbea36

```

47     constructor (
48         address gateway_,
49         address asset_,
50         address market_,
51         address comptroller_,
52         address rewardToken_
53     ) {
54         if (!IVenusMarket(market_).isVToken()) {
55             revert NotMarket();
56         }
57         if (asset_ != assetETH && IVenusMarket(market_).underlying() != asset_) {
58             revert InvalidAsset();
59         }
60
61         gateway = gateway_;

```

```

62     asset = asset_;
63     market = market_;
64     comptroller = comptroller_;
65     rewardToken = rewardToken_;
66 }

```

Listing 3.1: VaultImplementationVenus::constructor()

Recommendation Improve the validation of the given `market_` argument when interacting with the `vBNB` market.

Status This issue has been fixed in the following commit: [a23ba59](#).

3.2 Possible Precision Issue in DToken Redemption

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: Multiple Contracts
- Category: Numeric Errors [8]
- CWE subcategory: CWE-190 [1]

Description

The `Deri-V4` protocol has the built-in support of allowing LPs to provide funds into `vaults` for rewards. And LP share is accounted inside each individual `vault`. While reviewing current logic of minting and redeeming LP share, we notice a possible precision issue.

To elaborate, we show below the related `redeem()` routine. As the name indicates, this routine is designed to redeem LP share in exchange for the underlying asset. When the user indicates the underlying asset amount, the respective LP share is computed as `burnedSt = stTotal * redeemedAmount / amountTotal` (line 90). Unfortunately, the current approach may unintentionally introduce a precision issue by computing the `burnedSt` number against the protocol. Specifically, it takes a flooring-based division, which needs to be revised to take a ceiling-based division.

```

242     function redeem(uint256 dTokenId, uint256 amount) external _onlyGateway_ returns (
243         uint256 redeemedAmount) {
244         uint256 stAmount = stAmounts[dTokenId];
245         uint256 stTotal = stTotalAmount;
246
247         // Calculate the available assets ('available') for redemption based on staked
248         // amount ratios
249         uint256 amountTotal = asset.balanceOfThis();
250         uint256 available = amountTotal * stAmount / stTotal;
251         redeemedAmount = SafeMath.min(amount, available);

```

```

251     // Calculate the staked tokens burned ('burnedSt') based on changes in the total
        asset balance
252     uint256 burnedSt = stTotal * redeemedAmount / amountTotal;

254     // Update the staked amount for 'dTokenId' and the total staked amount
255     stAmounts[dTokenId] -= burnedSt;
256     stTotalAmount -= burnedSt;

258     asset.transferOut(gateway, redeemedAmount);
259 }

```

Listing 3.2: VaultImplementationNone::redeem()

Recommendation Properly revise the above routine to ensure the precision loss needs to be computed in favor of the protocol, instead of the user. Note that this issue affects a number of vaults.

Status This issue has been fixed in the following commit: [a23ba59](#).

3.3 Possible Costly DToken From Improper Vault Initialization

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: Multiple Contracts
- Category: Time and State [6]
- CWE subcategory: CWE-362 [3]

Description

As mentioned earlier, the Deri-V4 protocol allows users to deposit supported assets and get in return the share to represent the vault pool ownership. While examining the share calculation with the given deposits, we notice an issue that may unnecessarily make the pool share 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 supported assets and get respective pool shares in return. The issue occurs when the pool is being initialized under the assumption that the current pool is empty.

```

90     function deposit(uint256 dTokenId, uint256 amount) external payable _onlyGateway_
        returns (uint256 mintedSt) {
91         if (asset == assetETH) {
92             amount = msg.value;
93         } else {
94             asset.transferIn(gateway, amount);
95         }

```

```

97      // Calculate the 'mintedSt' based on the total staked amount ('stTotal') and the
          amount of assets deposited ('amount')
98      // 'mintedSt' is in 18 decimals
99      uint256 stTotal = stTotalAmount;
100     if (stTotal == 0) {
101         mintedSt = amount.rescale(asset.decimals(), 18);
102     } else {
103         uint256 amountTotal = asset.balanceOfThis();
104         mintedSt = stTotal * amount / (amountTotal - amount);
105     }

107     // Update the staked amount for 'dTokenId' and the total staked amount
108     stAmounts[dTokenId] += mintedSt;
109     stTotalAmount += mintedSt;
110 }

```

Listing 3.3: VaultImplementationNone::deposit()

Specifically, when the pool is being initialized (line 62), the share value directly takes the value of `mintedSt = amount.rescale(asset.decimals(), 18)` (line 63), which is manipulatable by the malicious actor. As this is the first deposit, the current total supply equals the calculated `stTotalAmount = mintedSt = 1 WEI`. With that, the actor can further deposit a huge amount of the underlying assets with the goal of making the pool share extremely expensive.

An extremely expensive pool share 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 computed pool 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.

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 share amount when the pool is being initialized. An alternative solution is to ensure a guarded launch process that safeguards the first deposit to avoid being manipulated.

Status The issue has been resolved by forcing a minimum initial minting share in the following commit: [a23ba59](#).

3.4 Trust Issue of Admin Keys

- ID: PVE-004
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: Multiple Contracts
- Category: Security Features [5]
- CWE subcategory: CWE-287 [2]

Description

In the Deri-V4 protocol, there is a special administrative account, i.e., `admin`, which plays a critical role in governing and regulating the system-wide operations (e.g., authorizing other roles, setting various parameters, and adjusting external oracles). It also has the privilege to regulate or govern the flow of assets among the involved components.

With great privilege comes great responsibility. Our analysis shows that the `admin` account is indeed privileged. In the following, we show representative privileged operations in the Deri-V4 protocol.

```

125     function setParameterOfId(string memory symbol, uint8 category, uint8 parameterId,
126         bytes32 value) external _onlyAdmin_ {
127         bytes32 symbolId = getSymbolId(symbol, category);
128         mapping(uint8 => bytes32) storage state = _states[symbolId];
129         if (category == CATEGORY_FUTURES) {
130             Futures.setParameterOfId(symbolId, state, parameterId, value);
131         } else if (category == CATEGORY_OPTION) {
132             Option.setParameterOfId(symbolId, state, parameterId, value);
133         } else if (category == CATEGORY_POWER) {
134             Power.setParameterOfId(symbolId, state, parameterId, value);
135         } else {
136             revert InvalidSymbolId(symbolId);
137         }
138     }
139
140     function setParameterOfIdForCategory(uint8 category, uint8 parameterId, bytes32
141         value) external _onlyAdmin_ {
142         uint256 length = _symbolIds.length();
143         for (uint256 i = 0; i < length; i++) {
144             bytes32 symbolId = _symbolIds.at(i);
145             if (getCategory(symbolId) == category) {
146                 mapping(uint8 => bytes32) storage state = _states[symbolId];
147                 if (category == CATEGORY_FUTURES) {
148                     Futures.setParameterOfId(symbolId, state, parameterId, value);
149                 } else if (category == CATEGORY_OPTION) {
150                     Option.setParameterOfId(symbolId, state, parameterId, value);
151                 } else if (category == CATEGORY_POWER) {
152                     Power.setParameterOfId(symbolId, state, parameterId, value);
153                 } else {

```

```
152         revert InvalidSymbolId(symbolId);
153     }
154 }
155 }
156 }
```

Listing 3.4: Example Privileged Operations in `SymbolManagerImplementation`

We emphasize that the privilege assignment with various core contracts is necessary and required for proper protocol operations. However, it would be worrisome if the `admin` is not governed by a DAO-like structure. The discussion with the team has confirmed that it is currently managed by a timelock contract with mandatory minimum 2 days cool down period. We point out that a compromised `admin` account would allow the attacker to undermine necessary assumptions behind the protocol and subvert various protocol operations.

Moreover, it should be noted that current contracts have the support of being deployed behind a proxy. And 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 with a `timelock` contract. Any admin operations will be executed through the `timelock` contract which imposes a mandatory minimum 1 day cool down period between submitting a transaction and executing it. The community can use this period to check the transaction and its possible consequence.

4 | Conclusion

In this audit, we have analyzed the `Deri-V4` protocol design and implementation. The `Deri` protocol is a decentralized protocol for users to exchange risk exposures precisely and capital-efficiently. The audited `Deri-V4` protocol is upgraded from `V3` with the adoption of the `Cross-Chain Decentralized Application (xDapp)` model for enhanced inclusivity, capital efficiency, and user experience. During the audit, we notice that the current code base is well organized and 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-190: Integer Overflow or Wraparound. <https://cwe.mitre.org/data/definitions/190.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-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: 7PK - Time and State. <https://cwe.mitre.org/data/definitions/361.html>.
- [7] MITRE. CWE CATEGORY: Business Logic Errors. <https://cwe.mitre.org/data/definitions/840.html>.
- [8] MITRE. CWE CATEGORY: Numeric Errors. <https://cwe.mitre.org/data/definitions/189.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.
- [11] PeckShield. PeckShield Inc. <https://www.peckshield.com>.

