SMART CONTRACT AUDIT REPORT

for

Deri Protocol (Aptos/Supra)

Prepared By: Xiaomi Huang

PeckShield

March 27, 2025

Document Properties

Client	Deri Protocol
Title	Smart Contract Audit Report
Target	Deri-V4
Version	1.0
Author	Xuxian Jiang
Auditors	Daisy Cao, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	March 27, 2025	Xuxian Jiang	Final Release
1.0-rc1	March 22, 2025	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 Deri-V4 (Aptos/Supra) 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. This audit focuses on its Aptos/Supra support. The basic information of the Deri protocol is as follows:

Table 1.1: Basic Information of The Deri Protocol

Item	Description
Name	Deri Protocol
Website	https://deri.io
Type	Move Smart Contract
Platform	Aptos/Supra
Audit Method	Whitebox
Latest Audit Report	March 27, 2025

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/dfactory-tech/deriprotocol-v4-supra.git (c7e5e3e)

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

- https://github.com/dfactory-tech/deriprotocol-v4-supra.git (fe29ada, bb6df72)

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

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 (Aptos/Supra)` 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	2	
Medium	7	
Low	2	
Informational	0	
Total	11	

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 high-severity vulnerabilities, 7 medium-severity vulnerabilities, and 2 low-severity vulnerabilities.

Table 2.1: Key Deri-V4 Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Improper i256::rescale() Logic	Coding Practices	Resolved
PVE-002	Medium	Improved Liquidity Addition Logic in gateway	Business Logic	Resolved
PVE-003	High	Incorrect transfer_out() Logic in gateway	Business Logic	Resolved
PVE-004	Medium	Arithmetic Underflow Avoidance in gateway	Numeric Errors	Resolved
PVE-005	Low	Incomplete FinishAddMargin Event Upon Margin Addition)	Coding Practices	Resolved
PVE-006	High	Incorrect Account Cache During Margin Removal	Business Logic	Resolved
PVE-007	Medium	Possibly Incorrect Removal Amount in Liquidity Update	Business Logic	Resolved
PVE-008	Low	Improved Admin Transition Logic in global_state	Coding Practices	Resolved
PVE-009	Medium	Trust Issue of Admin Keys	Security Features	Mitigated
PVE-010	Medium	Improper request_add_margin_b0() Logic in gateway	Business Logic	Resolved
PVE-011	Medium	Revisited check_b_token_consistency() Logic in gateway	Business Logic	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 Improper i256::rescale() Logic

- ID: PVE-001
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `i256`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The `Deri-V4` protocol has the built-in arithmetic operation support for unsigned integers (with the so-called `I256` type). While reviewing the `I256` type support, we notice one core `rescale()` routine has an incorrect implementation.

```
public fun rescale(num: I256, decimals_s1: u8, decimals_s2: u8): I256 {
    if (decimals_s1 == decimals_s2) {
        num
    } else {
        neg_from(
            abs_u256(num) *
                (math64::pow(10, (decimals_s2 as u64)) as u256) /
                (math64::pow(10, (decimals_s1 as u64)) as u256)
        )
    }
}
```

Listing 3.1: `i256::rescale()`

To elaborate, we show above the implementation of this specific `rescale()` routine. This routine has a rather straightforward logic in adjusting the given `I256 num` from the first decimal `decimals_s1` to the second decimal `decimals_s1`. When these two decimals are different, it always returns the negative number, which is apparently incorrect. A suggestion revision is shown as follows:

```
public fun rescale(num: I256, decimals_s1: u8, decimals_s2: u8): I256 {
    if (decimals_s1 == decimals_s2) {
```

```
            num
        } else {
            sign(num) == 1
            let rescaled_num = abs_u256(num) *
                    (math64::pow(10, (decimals_s2 as u64)) as u256) /
                    (math64::pow(10, (decimals_s1 as u64)) as u256);
            if (sign(num) == 1) {
                neg_from(rescaled)
            } else { rescaled }
        }
    }
```

Listing 3.2: Revised `i256::rescale()`

Recommendation Improve the above routine to properly adjust an `I256 num` from one decimal to another.

Status This issue has been fixed in the following commit: 92d6d18.

3.2 Improved Liquidity Addition Logic in gateway

- ID: PVE-002
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `gateway`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

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 issue in a corner case when handling the native coin addition (e.g., `SupraCoin`).

To elaborate, we show below the code snippet from the related `request_add_liquidity()` routine. As the name indicates, this routine is designed to add liquidity into the protocol vault in exchange for possible gains and rewards. When the user indicates the underlying asset amount in the native coin, the respective `b_amount` is always equal to `request_add_liquidity_fee`, which is incorrect. In fact, we need to withdraw `request_add_liquidity_fee + b_amount`, instead of `request_add_liquidity_fee` (line 842). Moreover, we need to avoid withdrawing from the user-specific `primary_fungible_store` (line 855) when the underlying asset amount is in the native coin.

```
842         let apt_fee_asset = coin_wrapper::wrap(coin::withdraw<SupraCoin>(user, (
                request_add_liquidity_fee as u64)));
843         let apt_amount = receive_execution_fee(
```

```
            d_token_state,
            gateway_state,
            gateway_param,
            request_add_liquidity_fee,
            apt_fee_asset
        );
        if (get_aptos_coin_wrapper() == b_token) {
            b_amount = apt_amount;
        };
        assert!(b_amount != 0, EINVALID_BTOKEN_AMOUNT);

        let b_token_asset = primary_fungible_store::withdraw(user, b_token, (b_amount as u64));
        deposit(&mut data, b_token_asset, gateway_param);
        get_ex_params(&mut data, b_token_state, gateway_param);
```

Listing 3.3: `gateway::request_add_liquidity()`

Recommendation Properly revise the above routine to ensure the native coin addition as liquidity is properly supported. An example (incomplete) revision of the above code snippet is shown as below:

```
        let apt_amount = if (get_aptos_coin_wrapper() == b_token) {
            receive_execution_fee + b_amount
        } else { receive_execution_fee }

        let apt_fee_asset = coin_wrapper::wrap(coin::withdraw<SupraCoin>(user, (apt_amount as u64)));
        receive_execution_fee(
            d_token_state,
            gateway_state,
            gateway_param,
            request_add_liquidity_fee,
            apt_fee_asset
        );

        assert!(b_amount != 0, EINVALID_BTOKEN_AMOUNT);

        if (get_aptos_coin_wrapper() != b_token) {
            let b_token_asset = primary_fungible_store::withdraw(user, b_token, (b_amount as u64));
            deposit(&mut data, b_token_asset, gateway_param);
        } else {
            /// TODO - deposit remaining native coins as well
        }
        get_ex_params(&mut data, b_token_state, gateway_param);
```

Listing 3.4: Revised `gateway::request_add_liquidity()`

Status This issue has been fixed in the following commit: 92d6d18.

3.3 Incorrect transfer_out() Logic in gateway

- ID: PVE-003
- Severity: High
- Likelihood: Medium
- Impact: High
- Target: gateway
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

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 logic to redeem their shares, we notice an issue that may incorrectly compute the amount for withdrawal.

To elaborate, we show below the code snippet from the transfer_out() routine, which is used for participating users to withdraw their liquidity or margins. The issue occurs when the vault has an insufficient balance, which results in the minting of IOU tokens. When IOU tokens are minted, the data.b0_amount state needs to computed as data.b0_amount = i256::sub(data.b0_amount, i256::add(i256::from(b0_out), i256::from(iou_amount))), not current data.b0_amount = i256::sub(data.b0_amount, i256::from(iou_amount)) (line 2055).

```
2023            if (amount > 0) {
2024                let b0_out;
2025                if (amount > b0_amount_in) {
2026                    // Redeem B0 tokens from vault0
2027                    let b0_redeemed_fungible_asset = vault::redeem(
2028                        object::address_to_object<Vault>(gateway_param.vault0),
2029                        0,
2030                        amount - b0_amount_in
2031                    );
2032                    let b0_redeemed = (fungible_asset::amount(&
                        b0_redeemed_fungible_asset) as u256);
2033                    fungible_asset::deposit(
2034                        get_gateway_store(gateway_param, gateway_param.token_b0).store,
2035                        b0_redeemed_fungible_asset
2036                    );

2038                    if (b0_redeemed < amount - b0_amount_in) {
2039                        // b0 insufficent
2040                        if (is_td) {
2041                            // Issue IOU for trader when B0 insufficent
2042                            iou_amount = amount - b0_amount_in - b0_redeemed;
2043                        } else {
2044                            // Revert for Lp when B0 insufficent
2045                            abort error::aborted(EINSUFFICIENT_B0_BALANCE)
2046                        }
2047                    };
```

```
                b0_out = b0_amount_in + b0_redeemed;
                b0_amount_in = 0;
            } else {
                b0_out = amount;
                b0_amount_in = b0_amount_in - amount;
            };
            b0_amount_out = b0_out;
            data.b0_amount = i256::sub(data.b0_amount, i256::from(iou_amount));
        };
```

Listing 3.5: `gateway::transfer_out()`

Recommendation Revise the above logic to properly compute the amount for withdrawal.

Status This issue has been fixed in the following commit: 92d6d18.

3.4 Arithmetic Underflow Avoidance in gateway

- ID: PVE-004
- Severity: Medium
- Likelihood: Low
- Impact: Medium
- Target: `gateway`
- Category: Numeric Errors [8]
- CWE subcategory: CWE-190 [2]

Description

The `Deri-V4` protocol allows users to flexibly manage their liquidity and margin. While reviewing current logic to remove the user liquidity, we notice the calculation to compute the remaining liquidity has a potential risk that may result in arithmetic underflow.

To elaborate, we show below the related `get_d_token_liquidity_with_remove_b0()` routine. As the name indicates, this routine is designed to calculate the liquidity if the given `bAmount` in `bToken` is removed. In particular, the internal variable `b0_total` is computed by directly making use of the arithmetic operation (lines 1871-1875), which should be guarded for possible overflows and underflows. While the overflow case is highly unlikely, the underflow case (line 1874) remains possible.

```
1862    fun get_d_token_liquidity_with_remove_b0(
1863        self: &Data, gateway_param: &GatewayParam, b0_amount_to_remove: u256
1864    ): u256 {
1865        let b_amount_in_vault =
1866            vault::get_balance(object::address_to_object<Vault>(self.vault), self.
                d_token_id);
1867        // discounted
1868        let b0_value_of_b_amount_in_vault =
1869            b_amount_in_vault * self.b_price / UONE * self.collateral_factor / UONE;
1870        let b0_total =
```

```
            if (!i256::is_neg(self.b0_amount)) {
                b0_value_of_b_amount_in_vault + i256::as_u256(self.b0_amount)
            } else {
                b0_value_of_b_amount_in_vault - i256::abs_u256(self.b0_amount)
            };

        if (b0_total > b0_amount_to_remove) {
            let decimals_b0 = fungible_asset::decimals(gateway_param.token_b0);
            safe_math256::rescale(b0_total - b0_amount_to_remove, decimals_b0,
                SCALE_DECIMALS)
        } else { 0 }
    }
```

Listing 3.6: `gateway::get_d_token_liquidity_with_remove_b0()`

Recommendation Properly revise the above routine to ensure the arithmetic overflow/underflow risk is completely eliminated.

Status This issue has been fixed in the following commit: 92d6d18.

3.5 Incomplete FinishAddMargin Event Upon Margin Addition

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `gateway`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

In `Aptos/Supra`, 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 `gateway` contract as an example. This contract has public functions that are used to adjust the margin for user positions. While examining the `FinishAddMargin` event that reflects user margin change, we notice this event is missing when a user requests to add margin in the base token (`b0`).

```
    public entry fun request_add_margin_b0(
        user: &signer,
        p_token_id: u256,
        b0_amount: u256
```

```
    ) acquires GatewayParam, GatewayStorage {
        let user_addr = signer::address_of(user);
        assert!(b0_amount > 0, EINVALID_BTOKEN_AMOUNT);
        check_p_token_id_owner(p_token_id, user_addr);
        let gateway_storage = borrow_global_mut<GatewayStorage>(@deri);
        let gateway_param = borrow_global<GatewayParam>(@deri);
        let token_b0 = gateway_param.token_b0;

        let b0_asset = primary_fungible_store::withdraw(user, token_b0, (b0_amount as
            u64));
        vault::deposit(
            object::address_to_object(gateway_param.vault0),
            p_token_id,
            b0_asset
        );

        let d_token_state = smart_table::borrow_mut(&mut gateway_storage.d_token_states,
             p_token_id);
        d_token_state.b0_amount = i256::wrapping_add(d_token_state.b0_amount, i256::from
            (b0_amount));
    }
```

Listing 3.7: `gateway::request_add_margin_b0()`

Recommendation Accurately emit the respective event when the user margin is adjusted.

Status This issue has been fixed in the following commit: fe29ada.

3.6 Incorrect Account Cache During Margin Removal

- ID: PVE-006
- Severity: High
- Likelihood: High
- Impact: High
- Target: `gateway`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

As mentioned earlier, the `Deri-V4` protocol has the built-in support of allowing traders to provide or withdraw margins. While reviewing current logic to withdraw the user margin, we notice a possible issue that uses the wrong account address for margin adjustment.

To elaborate, we show below the code snippet from the related `finish_remove_margin_internal()` routine. As the name indicates, this routine is designed to remove user margin. We notice the user account in current implementation is directly derived from the signing user `user`, which is incorrect.

The intended user account should be the owner of the given `p_token_id`, i.e., `ptoken::owner(p_token_id )` (line 1358).

```
1337     fun finish_remove_margin_internal(
1338         user: &signer,
1339         request_id: u256,
1340         p_token_id: u256,
1341         required_margin: u256,
1342         cumulative_pnl_on_engine: I256,
1343         b_amount_to_remove: u256
1344     ) acquires GatewayStorage, GatewayParam {
1345         let user_addr = signer::address_of(user);
1346         let gateway_storage = borrow_global_mut<GatewayStorage>(@deri);
1347         let gateway_param = borrow_global<GatewayParam>(@deri);

1349         let d_token_state = smart_table::borrow_mut(&mut gateway_storage.d_token_states,
              p_token_id);
1350         let b_token = d_token_state.b_token;
1351         let b_token_state = smart_table::borrow(&gateway_storage.b_token_states, object
             ::object_address(&b_token));

1353         check_request_id(d_token_state, request_id);
1354         let data = get_data_and_check_b_token_consistency(
1355             &gateway_storage.gateway_state,
1356             b_token_state,
1357             d_token_state,
1358             user_addr,
1359             p_token_id,
1360             b_token
1361         );
1362         ...
1363     }
```

Listing 3.8: `gateway::finish_remove_margin_internal()`

Recommendation Properly revise the above routine to ensure the correct user account address is used.

Status This issue has been fixed in the following commit: 92d6d18.

3.7 Possibly Incorrect Removal Amount in Liquidity Update

- ID: PVE-007
- Severity: Medium
- Likelihood: Medium
- Impact: High
- Target: `gateway`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The `Deri-V4` protocol has the essential logic in allowing users to withdraw their deposited liquidity. While examining the logic to redeem their liquidity, we notice an issue that may incorrectly finalize the withdrawal amount.

To elaborate, we show below the code snippet from the `finish_update_liquidity_internal()` routine, which is used for finalizing the user request of liquidity withdrawal. The internal `b0_amount_removed_asset` variable keeps track of the removed assets in `tokenB0` and the amount should be `safe_math256::min(b_amount_to_remove, i256::as_u256(data.b0_amount))` not current `safe_math256::min(b_amount_removed, i256::as_u256(data.b0_amount))` (line 1266).

```
1251            if (object::object_address(&data.b_token) == operate_token) {
1252                get_ex_params(&mut data, b_token_state, gateway_param);
1253                let transfer_out_amount = if (liquidity == 0) {
1254                    MAX_AS_U256
1255                } else {
1256                    b_amount_to_remove
1257                };
1258                b_amount_removed = transfer_out(&mut data, gateway_param,
                        transfer_out_amount, false);
1259            } else {
1260                assert!(operate_token == object::object_address(&gateway_param.token_b0)
                        , EINVALID_OPERATE_TOKEN);

1262                if (i256::is_greater_than_zero(data.b0_amount)) {
1263                    let b0_amount_removed_asset = vault::redeem(
1264                        object::address_to_object<Vault>(gateway_param.vault0),
1265                        0,
1266                        safe_math256::min(b_amount_removed, i256::as_u256(data.b0_amount
                                ))
1267                    );
1268                    let b0_amount_removed = (fungible_asset::amount(&
                            b0_amount_removed_asset) as u256);
1269                    data.b0_amount = i256::wrapping_sub(data.b0_amount, i256::from(
                            b0_amount_removed));

1271                    let b_amount_to_remove_asset = fungible_asset::extract(
1272                        &mut b0_amount_removed_asset,
1273                        (b_amount_to_remove as u64)
```

```
                    );
                    primary_fungible_store::deposit(data.account, b_amount_to_remove_asset);

                    let gateway_store = smart_table::borrow(&gateway_param.gateway_stores, gateway_param.token_b0);
                    fungible_asset::deposit(gateway_store.store, b0_amount_removed_asset);
                }
            }
```

Listing 3.9: `gateway::finish_update_liquidity_internal()`

Recommendation Revise the above logic to properly compute the amount for withdrawal.

Status This issue has been fixed in the following commit: 92d6d18.

3.8 Improved Admin Transition Logic in global_state

- ID: PVE-008
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `global_state`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

To facilitate the resource management, the `Deri-V4` protocol has a standalone `global_state` module, which defines a specific resource type `GlobalState`. This resource type defines the protocol-wide admin account as well as the pending admin for the intended two-phrase admin ownership transfer. Our analysis on the resource type indicates that the pending admin can be defined as `Option<address>`, not current `address`.

To elaborate, we show below the definition of this specific resource type. This resource type has three member fields: `extend_ref`, `admin`, and `pending_admin`. Among these three fields, the `pending_admin` field is better defined with the `Option<address>` type to indicate the possible absence when no admin ownership transfer is initiated.

```
    struct GlobalState has key {
        extend_ref: ExtendRef,
        admin: address,
        pending_admin: address
    }
```

Listing 3.10: The `GlobalState` Resource Type

Recommendation Properly revise the above resource type to better meet the intended design. The type change will also affect other related routines, including `init_module()`, `transfer_admin()` , and `accept_admin()`.

Status This issue has been fixed in the following commit: 92d6d18.

3.9 Trust Issue of Admin Keys

- ID: PVE-009
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `Multiple Contracts`
- Category: Security Features [5]
- CWE subcategory: CWE-287 [3]

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., create vaults, manage tokens, configure and collect protocol fees, as well as 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.

```
    public entry fun create_vault(admin: &signer, vault_asset: Object<Metadata>) {
        global_state::assert_is_admin(admin);
        vault::create_vault(vault_asset);
    }

    public entry fun add_b_token(
        admin: &signer,
        b_token: Object<Metadata>,
        vault_address: address,
        oracle_id: String,
        collateral_factor: u256
    ) acquires GatewayStorage, GatewayParam {
        global_state::assert_is_admin(admin);

        let gateway_storage = borrow_global_mut<GatewayStorage>(@deri);
        let gateway_param = borrow_global_mut<GatewayParam>(@deri);
        let b_token_states = &mut gateway_storage.b_token_states;
        ...
    }
    ...
    public entry fun set_b_token_parameter(
```

```
        admin: &signer,
        b_token: Object<Metadata>,
        vault_address: address,
        oracle_id: String,
        collateral_factor: u256
    ) acquires GatewayStorage {
        global_state::assert_is_admin(admin);
        ...
    }
    ...
    public entry fun set_execution_fee(
        admin: &signer,
        request_add_liquidity: u256,
        request_remove_liquidity: u256,
        request_remove_margin: u256,
        request_trade: u256,
        request_trade_and_remove_margin: u256
    ) acquires GatewayStorage {
        global_state::assert_is_admin(admin);
        ...
    }
    ...
```

Listing 3.11: Example Privileged Operations in `deri::gateway`

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. We point out that a compromised `admin` account would allow the attacker to undermine necessary assumptions behind the protocol and subvert various protocol operations.

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 mitigated with the use of a multi-sig account to hold the admin role.

3.10 Improper request_add_margin_b0() Logic in gateway

- ID: PVE-010
- Severity: Medium
- Likelihood: Medium
- Impact: High
- Target: `gateway`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The `Deri-V4` protocol also has the essential needs in allowing users to add margin for their positions. While examining the logic to add user margin in `tokenB0`, we notice an issue that may deposit into the wrong vault.

To elaborate, we show below the implementation of the affected `request_add_margin_b0()` routine. As the name indicates, this routine is used to top up the margin for a user position. With that, the user funds need to be deposited into the intended vault with credit to `d_token_id = 0`, not current `p_token_id` (line 969).

```
954     public entry fun request_add_margin_b0(
955         user: &signer,
956         p_token_id: u256,
957         b0_amount: u256
958     ) acquires GatewayParam, GatewayStorage {
959         let user_addr = signer::address_of(user);
960         assert!(b0_amount > 0, EINVALID_BTOKEN_AMOUNT);
961         check_p_token_id_owner(p_token_id, user_addr);
962         let gateway_storage = borrow_global_mut<GatewayStorage>(@deri);
963         let gateway_param = borrow_global<GatewayParam>(@deri);
964         let token_b0 = gateway_param.token_b0;

966         let b0_asset = primary_fungible_store::withdraw(user, token_b0, (b0_amount as
                u64));
967         vault::deposit(
968             object::address_to_object(gateway_param.vault0),
969             p_token_id,
970             b0_asset
971         );

973         let d_token_state = smart_table::borrow_mut(&mut gateway_storage.d_token_states,
                 p_token_id);
974         d_token_state.b0_amount = i256::wrapping_add(d_token_state.b0_amount, i256::from
                (b0_amount));
975     }
```

Listing 3.12: `gateway::request_add_margin_b0()`

Recommendation Revise the above logic to properly deposit into the intended vault.

Status This issue has been fixed in the following commit: 92d6d18.

3.11 Revisited check_b_token_consistency() Logic in gateway

- ID: PVE-011
- Severity: Medium
- Likelihood: Medium
- Impact: High
- Target: gateway
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

For gas efficiency and code management, the `gateway` contract in `Deri-V4` maintains a local data cache that has the common need of validating the `tokenB` for synchronization. While examining the related data cache validation logic, we notice an issue that should be fixed.

To elaborate, we show below the implementation of the related `check_b_token_consistency()` routine. As the name indicates, it is used to validate the `tokenB` consistency. By design, the `tokenB` may be changed only when the previous `tokenB`, if any, does not have any remaining balance. With that, the zero-balance validation should be performed on the vault associated with the previous `tokenB`, not current one (line 1712).

```
1703     fun check_b_token_consistency(
1704         d_token_state: &DTokenState,
1705         b_token_state: &BTokenState,
1706         d_token_id: u256,
1707         b_token: Object<Metadata>
1708     ) {
1709         let pre_b_token = d_token_state.b_token;
1710         let pre_b_token_addr = object::object_address(&pre_b_token);
1711         if (pre_b_token_addr != ZERO_ADDRESS && pre_b_token != b_token) {
1712             let vault_address = b_token_state.vault;

1714             let st_amount = vault::st_amounts(object::address_to_object(vault_address),
                     d_token_id);
1715             assert!(st_amount == 0, EINVALID_BTOKEN);
1716         }
1717     }
```

Listing 3.13: `gateway::check_b_token_consistency()`

Recommendation Revise the above logic to properly validate the `tokenB` consistency.

Status This issue has been fixed in the following commit: 92d6d18.

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. This audit focuses on its `Aptos/Supra` support. 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-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-190: Integer Overflow or Wraparound. https://cwe.mitre.org/data/definitions/190.html.

[3] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.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: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.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.