Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

HiYield

Veridise Inc.

May 29, 2024

► **Prepared For:**

Lydia Labs
lydialabs.xyz

► **Prepared By:**

Alberto Gonzalez
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

May 23, 2024	Draft Report.
May 29, 2024	Official Report.

Contents

1 Executive Summary

From Apr. 15, 2024 to May 15, 2024, Lydia Labs engaged Veridise to review the security of their HiYield project. The review covered HiYield - a suite of two lending protocols targeting traditional capital markets. Veridise conducted the assessment over 9 person-weeks, with 2 engineers reviewing code over 4.5 weeks on commit `1e0f9af`. The security assessment process involved extensive manual auditing.

Project summary. The main product of the protocol is the Credit Vault, which is reserved exclusively for whitelisted investors. These investors deposit stablecoin tokens that the admin can borrow at a fixed interest rate, receiving share tokens in return. It is expected that these shares will be converted back into a greater amount of stablecoin tokens than originally deposited, reflecting the accrued interest. It is important to note that the admin borrows funds without posting any collateral.

Code assessment. The developers of HiYield provided the source code of the HiYield contracts for review. The source code appears to be mostly original, programmed by the HiYield developers, except for the token contract, which was sourced from the Soroban SDK. To aid the Veridise auditors in understanding the code, the HiYield developers also shared project documentation. This included a high-level product overview and a detailed overview of smart-contract functions and storage variables. While the documentation was somewhat outdated compared to the latest product implementation, the underlying concepts remained largely relevant.

The source code for the credit vault contract included a test suite, which the Veridise auditors found useful for understanding common product use cases and for developing coded proofs of concept. However, they noted that certain contracts, such as the ai_pool, isolated_pool, and passive_pool, lacked a test suite.

Summary of issues detected. The audit uncovered 38 issues, 5 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, the issues V-HYD-VUL-001 V-HYD-VUL-002 could cause the Credit Vault to enter a denial of service state, and the issue V-HYD-VUL-005 could result in the deployment of outdated and potentially incorrect smart contracts to the live network. In addition the Veridise auditors also identified 4 medium-severity issues, including the issue V-HYD-VUL-006 that could cause incorrect interest calculation for users of the Credit Vault. Moreover, the Veridise auditors flagged 15 low issues, 9 warnings and 5 informational findings. Among these, the issue V-HYD-VUL-022 flags the centralization risks of the project.

Lydia Labs has fixed all the identified issues related to the Credit Vault product. The remaining issues reside in contracts not associated with the Credit Vault. Veridise auditors have been notified these contracts will not be deployed.

Recommendations. After auditing the code, the auditors had a few suggestions to improve the HiYield security:

- Given the relatively high level of centralization in the project, it is highly recommended to implement best practices for managing cryptographic keys to reduce the risk of key compromise.
- Modify the build process to ensure the repository does not contain stale WASM binaries and is self-sufficient. Currently, the source code cannot be built from scratch without extra manual effort.
- Clearly comment on the used token scale and/or measurement for each variable used in the financial calculations.

In addition, Veridise auditors recommend not using the codebase outside of the Credit Vault product in production, as previously mentioned, due to the lack of a test suite and the presence of 'TODOs' blocks in the code.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
HiYield	`1e0f9af`	Rust	Soroban

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Apr. 15 - May 15, 2024	Manual	2	9 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	1	1	1
High-Severity Issues	4	2	4
Medium-Severity Issues	4	2	4
Low-Severity Issues	15	7	15
Warning-Severity Issues	9	2	9
Informational-Severity Issues	5	3	5
TOTAL	38	17	38

Table 2.4: Category Breakdown.

Name	Number
Maintainability	11
Logic Error	9
Data Validation	9
Denial of Service	3
Usability Issue	2
Access Control	1
Centralization Risk	1
Invalid Interface	1
Missing/Incorrect Events	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of HiYield's smart contracts. In our audit, we sought to answer questions such as:

- Are lending protocols vulnerable to certain types of attacks, such as the First Depositor Attack or the Donation Attack?
- Is there a way for a malicious user to stop the protocol operations?
- Is there a way for a malicious user to gain financial advantage over other honest users?
- Is the implementation of the protocol consistent with the higher level business intent?
- Are there any platform-related implementation defects present in the project source code?
- For contracts that are intended for reuse in other projects, are there any assumptions that might differ in different contexts and could potentially lead to contract issues?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions raised above, we conducted a thorough manual review of the source code and prototyped proof-of-concept samples for confirming some issues.

Scope. Originally, the scope of this audit included all the *contracts* folder of the source code provided by the HiYield developers. During the audit process, project representatives requested that we prioritize the review based on their business needs. We agreed that some contracts would fall outside the scope of the review, while others would receive more attention.

The following contracts have undergone a full security review:

- Credit Vault
- Security Token
- Access Control
- Compliance
- Token
- Rewards
- Passive Pool
- Pause
- AI Pool

The following contract has undergone a partial review:

- Isolated Pool

The following contracts have not been reviewed:

- Controller

- Jump Rate Model
- Mock Isolated Pool
- Oracle
- StandAlone Pool

Methodology. Veridise auditors inspected the provided tests, and read the HiYield documentation. They then began a manual audit of the code. All identified issues were cross-checked by security engineers involved in the audit process. During the audit, the Veridise auditors had several meetings with HiYield product owner to ask questions about their business logic.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-HYD-VUL-001	Operations can be disrupted via the withdrawal . .	Critical	Fixed
V-HYD-VUL-002	Linked list is saved into instance storage	High	Fixed
V-HYD-VUL-003	Incorrect assumption about isolated pool shares. . .	High	Acknowledged
V-HYD-VUL-004	Pause and unpause incorrect implementations	High	Acknowledged
V-HYD-VUL-005	Stale WASM dependencies	High	Fixed
V-HYD-VUL-006	Interest might accrue incorrectly	Medium	Fixed
V-HYD-VUL-007	Borrow uses a stale value for total_deposits	Medium	Acknowledged
V-HYD-VUL-008	Lack of validation for minted shares in Passive. . .	Medium	Acknowledged
V-HYD-VUL-009	Insufficient token decimal value check	Medium	Fixed
V-HYD-VUL-010	Missing accrue interest before queue process	Low	Fixed
V-HYD-VUL-011	Incorrect token balance check	Low	Fixed
V-HYD-VUL-012	Total assets limit can be bypassed	Low	Fixed
V-HYD-VUL-013	Insufficient access restriction for functions r. . .	Low	Acknowledged
V-HYD-VUL-014	Insufficient argument check for the ltv variable	Low	Acknowledged
V-HYD-VUL-015	Incorrect value returned by get_info_by_address	Low	Acknowledged
V-HYD-VUL-016	Malicious user can manipulate the initial excha. . .	Low	Acknowledged
V-HYD-VUL-017	Linked_list operations are not checked for success	Low	Fixed
V-HYD-VUL-018	Credit Vault cannot make admin actions over sec. .	Low	Fixed
V-HYD-VUL-019	Role should default to admin role during initia. . .	Low	Acknowledged
V-HYD-VUL-020	Utilization rate will be computed incorrectly d. . .	Low	Acknowledged
V-HYD-VUL-021	Input sanitization best practices	Low	Fixed
V-HYD-VUL-022	Centralization Risk	Low	Acknowledged
V-HYD-VUL-023	Insufficient number of parameters supplied to t. . .	Low	Fixed
V-HYD-VUL-024	Insufficient input validation in claim_rewards . . .	Low	Acknowledged
V-HYD-VUL-025	Scale value is not tied to token decimals	Warning	Acknowledged
V-HYD-VUL-026	Hard assumption around token decimals	Warning	Fixed
V-HYD-VUL-027	Inconsistency on interest computation	Warning	Fixed
V-HYD-VUL-028	Inconsistency on the check for the max_utilizat. . .	Warning	Acknowledged
V-HYD-VUL-029	pool_info is not implementing the interface cor. . .	Warning	Acknowledged
V-HYD-VUL-030	Issues when using declared interfaces	Warning	Acknowledged
V-HYD-VUL-031	Missing event on state update	Warning	Acknowledged
V-HYD-VUL-032	AccessControl implementation lacks function to . .	Warning	Acknowledged
V-HYD-VUL-033	Access control roles should be saved into persi. . .	Warning	Acknowledged
V-HYD-VUL-034	Duplicated code	Info	Fixed
V-HYD-VUL-035	Pending withdraw assets variable lacks external. . .	Info	Fixed
V-HYD-VUL-036	Code repetition in several places	Info	Acknowledged
V-HYD-VUL-037	Documentation Improvements	Info	Acknowledged
V-HYD-VUL-038	The linked list size should not be allowed to c. . .	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-HYD-VUL-001: Operations can be disrupted via the withdrawal queue

Severity	Critical	**Commit**	1e0f9af
Type	Denial of Service	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	queue_process		
Confirmed Fix At	N/A		

Withdrawals are processed via a queue, and almost every action in the contract, such as deposits and debt repayments, triggers the `_queue_process` function. This function is responsible for processing withdrawals in a FIFO order. It will halt with a panic if it encounters a request with zero or a negative asset amount.

However, there is a vulnerability: a malicious user could exploit this behavior by introducing a withdrawal request for zero assets into the queue. This scenario is not validated during the withdrawal execution flow. To demonstrate this issue, the following proof of concept can be added to the `test` file under the `credit_vault` directory.

```
#[test]
#[should_panic(expected= "CV: AMOUNT_MUST_NOT_ZERO")]
fn test_brick_queue() {
    let e = Env::default();
    e.mock_all_auths();
    e.budget().reset_unlimited();
    let (admin1, user1, user2, user3, usdc, credit_vault, debt_token, _) = set_up(&e)
    ;

    // User1 deposits 1000 assets
    credit_vault.deposit(&user1, &1000_0000000, &user1);
    // Admin borrows the previously deposited funds
    credit_vault.borrow_fund(&admin1, &1000_0000000);
    // User1 requests a withdrawal.
    credit_vault.withdraw(&user1, &50_0000000, &user1, &user1);

    // User2 disrupts the queue process by requesting a withdrawal of 0 assets
    credit_vault.deposit(&user2, &2, &user2);
    credit_vault.withdraw(&user2, &0, &user2, &user2);

    // Subsequent withdrawals are affected
    credit_vault.withdraw(&user1, &50_0000000, &user1, &user1);

    // User3 deposits sufficient funds to process previous withdrawals, but the
    process fails.
    credit_vault.deposit(&user3, &1050_0000000, &user3);
}
```

Snippet 4.1: Proof of concept that demonstrates a denial of service (DoS) vulnerability in the withdrawal queue.

Impact This vulnerability can result in a denial of service (DoS) for most of the contract's main functions since they invoke `_process_queue` at the end.

Recommendation To mitigate this issue, the contract should disallow withdrawals of zero assets and zero shares, adopting a validation approach similar to that used in the deposit flow.

Developer Response Recommendation implemented.

4.1.2 V-HYD-VUL-002: Linked list is saved into instance storage

Severity	High	**Commit**	1e0f9af
Type	Denial of Service	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	write_node		
Confirmed Fix At	N/A		

The withdrawal queue in the `credit_vault` contract is structured using the `linked_list` logic. Each time a withdrawal request is initiated, it is appended to the linked list. Currently, list elements that make up this linked list are stored in the instance storage of the contract, rather than in persistent storage.

```
pub fn write_node(e: &Env, _node: u128, _pos: bool, _value: u128) {
    let key = LinkedListDataKey::List(ListDataKey { _node, _pos});
    e.storage().instance().set(&key, &_value);
}
```

Snippet 4.2: Code snippet from the `linked_list` file showing how elements are stored in instance storage.

Using instance storage, which has limited capacity, for unbounded data like withdrawal requests is prone to denial of service issues. Furthermore, instance storage is accessed during every interaction with the contract, which increases resource consumption and operational costs for users.

Soroban SDK reference.

Impact The limited capacity of instance storage can be exploited by malicious users who request numerous withdrawals, thereby preventing the addition of new withdrawal requests to the `credit_vault` contract. This limitation also leads to increased costs for users interacting with the contract, as all the instance storage is accessed every time the contract is invoked.

Recommendation It is advisable to utilize persistent storage for storing the linked list elements.

Developer Response Recommendation implemented.

4.1.3 V-HYD-VUL-003: Incorrect assumption about isolated pool shares balance

Severity	High	**Commit**	1e0f9af
Type	Logic Error	**Status**	Acknowledged
File(s)	`passive_pool/contract.rs`		
Location(s)	remove_pool, unwind_redeem		
Confirmed Fix At	N/A		

The `passive_pool` contract acts as a lender to various Isolated Pools but is not treated as a normal depositor. Unlike other depositors who receive minted shares from the Isolated Pool, the Passive Pool's shares of the Isolated Pool are tracked internally, as illustrated in the following code snippet:

```
write_type_i128(e, ISDataKey::PassivePoolBalance, pp_bal + (borrow_amount *
    SCALE_VALUE / exchange_rate));
```

Snippet 4.3: Code snippet from the `check_and_request_borrow_to_pp` function in the `isolated_pool` contract.

However, the functions `remove_pool` and `unwind_redeem` in the `passive_pool` contract attempt to redeem shares from the Isolated Pool only if the share balance is greater than zero, which will never be the case since the Isolated Pool does not mint shares for the Passive Pool.

```
// @audit-issue Passive pool does not hold shares.
let pp_bal = token::Client::new(&e, &pool_client.token_address()).balance(&e.
    current_contract_address());

if pp_bal > 0 {
    // revert if sub pool has not enough cash to cover redeem action
    pool_client.redeem(&e.current_contract_address(), &pp_bal);
}
```

Snippet 4.4: Code snippet from the `remove_pool` function.

Impact The Passive Pool is unable to redeem their shares back into stablecoin tokens due to the absence of minted shares.

Recommendation To resolve this, the `passive_pool` contract should query the internal shares balance tracked under the `ISDataKey::PassivePoolBalance` storage key in the `isolated_pool` contract instead of the balance in the share's token contract.

Developer Response

4.1.4 V-HYD-VUL-004: Pause and unpause incorrect implementations

Severity	High	**Commit**	1e0f9af
Type	Logic Error	**Status**	Acknowledged
File(s)	`pause/lib.rs`		
Location(s)	pause, unpause		
Confirmed Fix At	N/A		

The checks in the `pause` and `unpause` functions of the `pause` contract are mistakenly inverted. The current implementation panics under the wrong conditions, as illustrated in the provided code snippets:

```
pub fn pause(&self) {
   if !read_pause(&self.env) {
      panic!("paused");
   }
}

pub fn unpause(&self) {
   if read_pause(&self.env) {
       panic!("not paused");
   }
}
```

Snippet 4.5: Code snippets from `lib` file of the `Pause` crate.

Impact Due to this inversion, the `pause` function panics if the contract is not already paused, and the `unpause` function panics if the contract is paused, which contradicts their intended behavior.

Recommendation The logic in these functions should be corrected as follows:

- The `pause` function should check if the contract is already paused and only proceed if it is not.
- The `unpause` function should check if the contract is paused and only proceed if it is.

Developer Response

4.1.5 V-HYD-VUL-005: Stale WASM dependencies

Severity	High	**Commit**	1e0f9af
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

The build process for the project's smart contracts, specifically illustrated by the `build_release.sh` script, highlights a sequential build and optimization pattern.

```
#!/bin/bash

# build
soroban contract build

# Optimized builds
soroban contract optimize --wasm target/wasm32-unknown-unknown/release/ai_pool.wasm
soroban contract optimize --wasm target/wasm32-unknown-unknown/release/compliance.
    wasm
soroban contract optimize --wasm target/wasm32-unknown-unknown/release/controller.
    wasm
# and so on...
```

Snippet 4.6: Code snippet from the `build_realse.sh` file:

All contracts are compiled in alphabetical order, and optimizations are subsequently applied. This routine affects interdependencies between contracts, particularly when one contract imports another's WASM file. For example, `security_token.rs` in `credit_vault`:

```
soroban_sdk::contractimport!(
    file = "../../target/wasm32-unknown-unknown/release/security_token.optimized.wasm
    "
);
```

Snippet 4.7: Code snippet from `credit_vault/security_token.rs`

Here, the `credit_vault` contract is compiled with an already existing but potentially outdated version of `security_token.optimized.wasm` due to the alphabetical build order. This means if there are updates in the `security_token`, `credit_vault` may still compile against the old WASM, leading to inconsistencies and potentially incorrect implementations, which might not be caught even in tests as they are built after all other modules have been compiled and optimized. This issue is related to many contracts in this project and seems to be a common problem with Soroban's development process at the moment.

Impact This build process introduces a significant risk of deploying contracts that are not up-to-date with their dependencies, potentially leading to malfunctioning contracts being used in production. The risk is compounded by the fact that successful test results might mask the underlying issues, giving a false sense of security.

Recommendation To mitigate these risks, it's crucial to ensure that all contracts are built and optimized in an order that respects their dependencies. One strategy is to manually manage the build order in the script to ensure that dependencies are always built and optimized before the contracts that rely on them.

Additionally, it is advised to read a new article on this topic located here:

Stale WASM dependencies article.

Developer Response Recommendation was implemented.

4.1.6 V-HYD-VUL-006: Interest might accrue incorrectly

Severity	Medium	**Commit**	1e0f9af
Type	Logic Error	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	interest_assets		
Confirmed Fix At	N/A		

The interest accrued in the `credit_vault` smart contract is calculated based on the time elapsed since the last accrual, the borrowed assets, and the interest rate. Crucially, the computation makes a correction to the borrowed assets to account for the withdrawal requests by subtracting `pending_withdraw_assets` from `total_borrowed_assets` to reflect the actual amount of assets that should accrue interest.

The assumption here is that the difference `total_borrowed_assets - pending_withdraw_assets` accurately represents the real borrowed assets accruing interest. However, this will not always be the case when the withdrawal queue cannot be processed entirely when a debt repayment occurs.

To illustrate this issue, consider the following proof of concept, which can be added to the `test` file under the `credit_vault` directory. It demonstrates a scenario where the computed interest may turn negative:

Impact Interest calculations will be underestimated in scenarios where the withdrawal queue cannot be processed completely in a single run.

Recommendation A simple mitigation may not suffice without impacting other contract functionalities. For a long-term solution, a re-architecture of the accounting processes related to "real" borrowed funds is recommended.

In the short term, constant monitoring of the protocol is advisable to ensure that the withdrawal queue does not exceed the processing limit.

Developer Response Now, the code checks if the contract has enough assets to meet the pending withdrawals before subtracting them from the total borrowed assets.

```rust
#[test]
fn test_interest_inconsistency_repay_debt() {
    let e = Env::default();
    e.budget().reset_unlimited();
    e.mock_all_auths();
    let (admin1, user1, user2, user3, usdc, credit_vault, debt_token, _) = set_up(&e)
    ;

    // User2 deposits 1000 assets
    assert_eq!(credit_vault.deposit(&user2, &1000_0000000, &user2), 1000_0000000);
    // Admin borrows all the funds
    credit_vault.borrow_fund(&admin1, &1000_0000000);
    assert_eq!(credit_vault.total_borrowed_assets(), 1000_0000000);

    // Simulate 11 withdrawals. Totalling 990 assets.
    for _ in 0..11 {
        assert_eq!(credit_vault.withdraw(&user2, &90_0000000, &user2, &user2), 90
    _0000000);
    }

    // User1 deposits 50 assets
    assert_eq!(credit_vault.deposit(&user1, &50_0000000, &user1), 50_0000000);

    // Expected total assets is 60 after accounting for deposits and withdrawals
    // 1000 + 50 - 990.
    assert_eq!(credit_vault.total_assets(), 60_0000000);

    // Admin repays part of the borrowed funds
    credit_vault.repay_debt(&admin1, &940_0000000);

    // If we advance time, we would expect an increase in interest for the still 10
    borrowed assets.
    // Therefore, total_assets should exceed 60 afterwards.
    set_timestamp(&e, e.ledger().timestamp() + 100);

    // Resulting total assets are less than expected due to negative interest
    calculation
    assert!(credit_vault.total_assets() < 60_0000000);
}
```

Snippet 4.8: Proof of concept that demonstrates the computed interest becoming negative.

4.1.7 V-HYD-VUL-007: Borrow uses a stale value for total_deposits

Severity	Medium	Commit	1e0f9af
Type	Logic Error	Status	Acknowledged
File(s)	passive_pool/contract.rs		
Location(s)	borrow, total_deposit		
Confirmed Fix At	N/A		

The total deposits in the `passive_pool` contract are computed by summing the `cash` within the pool, the `loans` made to the isolated pool, and any interest accrued from those loans. This value is updated during the `_exchange_rate` function, as shown in the following snippet:

```
fn _exchange_rate(e: &Env) -> i128 {
    let total_supply = token::Client::new(&e, &read_token_contract(&e)).total_supply
    ();
    let cash = token::Client::new(&e, &Address::from_string(&String::from_str(e,
    UNDERLYING_TOKEN))).balance(&e.current_contract_address());
    let total_loaned_and_cash = _get_loaned_and_cash(&e, e.current_contract_address()
    , 0, cash);
    // Update total deposit
    write_total_deposit(&e, total_loaned_and_cash);

    __exchange_rate(total_supply, total_loaned_and_cash)
}
```

Snippet 4.9: `_exchange_rate` function from the `contract` file in passive pool.

However, certain parts of the code use a stale value of `total_deposits`. Notably, this occurs during the `borrow` function when calling the `_get_cash_prior` function and in the `total_deposit` external getter function.

```
if borrow_amount > _get_cash_prior(&e, cash, read_total_deposit(&e)) {
   panic!("PP: insufficient cash");
}
```

Snippet 4.10: Code snippet from the `borrow` function. The function `_get_cash_prior` is being passed a a up `total_deposit` value that is not up to date.

Impact The contract will read a stale value for the `total_deposits` variable. This will cause the contract to underestimate the `cash` available for borrowing to Isolated Pools.

Recommendation Ensure the value of `total_deposits` is updated before reading it in `borrow` and `total_deposit` functions.

Developer Response

4.1.8 V-HYD-VUL-008: Lack of validation for minted shares in Passive Pool

Severity	Medium	**Commit**	1e0f9af
Type	Data Validation	**Status**	Acknowledged
File(s)	`passive_pool/contract.rs`		
Location(s)	supply		
Confirmed Fix At	N/A		

In the `supply` function of the `passive_pool` contract, the code mistakenly validates twice that the `amount` of assets supplied is greater than zero. However, the issue lies in the fact that the second validation should actually check if the `mint_amount` is greater than zero, as highlighted in the code snippet below:

```
if amount <= 0 {
     panic!("PP: mint zero value");
}
```

Snippet 4.11: Code snippet from the `supply` function of the passive pool contract.

Impact Failing to validate that the `mint_amount` is greater than zero allows the possibility of so called First Depositor Attacks, when combined with the ability to manipulate the exchange rate as described in V-HYD-VUL-016. Here's a breakdown of how this attack works:

1. **Reserve Manipulation**: A malicious user sets up the pool so that both the reserves of assets and shares are `1`.
2. **Supply by Honest User**: An honest user attempts to supply `1000` assets, expecting to receive `1000` shares in return, based on the initial reserve setup.
3. **Intervention by Malicious User**: Before the honest user's transaction is processed, the attacker donates `1000` assets to the pool.

4. Minting Calculation: When the honest user's transaction is processed, the minting of shares is calculated as:

- `(1000 * 1e10) / (1001 * 1e10)` which is equal to zero.

1. **Result**: The honest user ends up supplying `1000` assets but receives no shares in return.
2. **Exploitation**: The malicious user then redeems their single share, which now represents all the assets in the pool.

Recommendation Change `amount <= 0` to `mint_amount <= 0`.

Developer Response

4.1.9 V-HYD-VUL-009: Insufficient token decimal value check

Severity	Medium	**Commit**	1e0f9af
Type	Data Validation	**Status**	Fixed
File(s)	`token/contract.rs`, `security_token/contract.rs`		
Location(s)	initialize		
Confirmed Fix At	N/A		

The current `token` and `security_token` implementation allows decimal values up to `u8::MAX`, which is incorrect since the type `i128` can hold only up to 38 digits (base 10). The decimal value should be chosen with a greater caution.

```
if decimal > u8::MAX.into() {
    panic!("Decimal must fit in a u8");
}
```

Snippet 4.12: Code snippet from `security_token::initialize()`.

Impact In the context of `credit_vault` and other pool contracts, this issue is not significant, since decimal value is set to `7`. However, since it is expected that token contracts are candidates for using in other projects and contexts, this issue may severely affect other projects.

Since the balance type of the token is `i128`, decimal value should be taken with a greater caution. For values larger than `18`, there is a significant probability of sporadic overflows in the token. It may severely affect the business logic of a protocol, up to being completely broken.

Recommendation It is recommended not to allow decimal value be greater than `18`.

Developer Response Recommendation was implemented.

4.1.10 V-HYD-VUL-010: Missing accrue interest before queue process

Severity	Low	**Commit**	1e0f9af
Type	Logic Error	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	queue_process		
Confirmed Fix At	N/A		

One of the key variables influencing the computation of accrued interest is `pending_withdraw_assets` . As such, it is crucial that `accrued_interest` is called before any modifications to this variable to ensure accuracy. Currently, the internal function `_queue_process`, which may alter `pending_withdraw_assets`, can be triggered via `queue_process` without a preceding call to `accrued_interest`.

Impact This sequence results in `accrued_interest` potentially using an outdated value for `pending_withdraw_assets` when it is finally called.

Recommendation To rectify this, it is advisable to invoke `accrued_interest` before `_queue_process` .

Note: Implementing this recommendation might increase the significance of the existing issue V-HYD-VUL-006 . If both issues are not simultaneously addressed, it is essential to evaluate again.

Developer Response Recommendation implemented.

4.1.11 V-HYD-VUL-011: Incorrect token balance check

Severity	Low	**Commit**	1e0f9af
Type	Logic Error	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	initialize		
Confirmed Fix At	N/A		

The `initialize` function of the `credit_vault` contract includes a branch where it utilizes `init_data` to set up the contract, for example minting a designated amount of shares to a list of accounts. To ensure that shares are not minted multiple times to the same account, the function includes a validation step checking that the account balances are zero before proceeding. However, the function currently uses the `asset_client` to verify balances, which is incorrect as it does not represent the vault shares, but rather the underlying deposited token.

Impact The incorrect check will not prevent double accounting mistakes.

Recommendation The balance check should be on `token_client` instead of `asset_client`.

Developer Response Recommendation was implemented.

4.1.12 V-HYD-VUL-012: Total assets limit can be bypassed

Severity	Low	**Commit**	1e0f9af
Type	Data Validation	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	cancel_withdraw		
Confirmed Fix At	N/A		

The vault maintains a variable `total_assets` which is capped by `max_total_assets` to control the total quantity of assets deposited within the vault. This cap is enforced in functions like `_deposit` to prevent exceeding the maximum allowable assets. However, the `_cancel_withdrawal` function, which increases the value of `total_assets`, currently lacks this validation.

Impact Without the check in `_cancel_withdrawal`, it is possible to circumvent the asset limit set by `max_total_assets`.

Recommendation It is advisable to implement the same asset cap in the `_cancel_withdrawal` function as is done in `_deposit`.

Developer Response Recommendation implemented.

4.1.13 V-HYD-VUL-013: Insufficient access restriction for functions repay_debt and add_reserves

Severity	Low	**Commit**	1e0f9af
Type	Access Control	**Status**	Acknowledged
File(s)	`ai_pool/contract.rs`		
Location(s)	repay_debt, add_reserves		
Confirmed Fix At	N/A		

The documentation for the AI Pool specifies that "*In order to repay debt to the AI Pool, the admin calls repayDebt and transfers an amount of stablecoins to the AI Pool.*"

However, it has been noted that there is no access control check in the `repay_debt` function of the AI Pool and the `add_reserves` function of the Isolated Pool. While allowing any user to contribute might seem harmless, potentially framing it as a donation, such openness can inadvertently become a vector for attacks.

Impact Allowing unrestricted access to functions intended for administrative tasks such as `repay_debt` and `add_reserves` may result in a wider attack surface.

Recommendation To mitigate potential security risks and align with the documented role permissions, restrict access to the `repay_debt` and `add_reserves` functions to the admin account only.

Developer Response

4.1.14 V-HYD-VUL-014: Insufficient argument check for the ltv variable

Severity	Low	**Commit**	1e0f9af
Type	Data Validation	**Status**	Acknowledged
File(s)	`isolated_pool/contract.rs`		
Location(s)	set_collateral, set_ltv		
Confirmed Fix At	N/A		

The function `set_ltv` in the current implementation only verifies that the Loan-to-Value (LTV) ratio is non-negative, missing a crucial check to ensure that LTV does not exceed 100%. Additionally, the function `set_collateral_state`, which also has the capability to modify the LTV, lacks any validation on the `ltv` argument.

Impact Setting an LTV ratio higher than 100% can disrupt the fundamental logic of borrowing, potentially leading to situations where borrowers could be permitted to take out loans of greater value than the collateral posted.

Recommendation It is advisable to add checks in both `set_ltv` and `set_collateral_state` functions to confirm that the LTV ratio is not only non-negative but also does not exceed 100%.

Developer Response

4.1.15 V-HYD-VUL-015: Incorrect value returned by get_info_by_address

Severity	Low	**Commit**	1e0f9af
Type	Logic Error	**Status**	Acknowledged
File(s)	`ai_pool/contract.rs`		
Location(s)	get_info_by_address		
Confirmed Fix At	N/A		

In the function `get_info_by_address`, the code returns two values: the user's security token balance and the amount of stable coin tokens those security tokens are worth. However, the computation of the latter value is incorrect:

```
1 get_ratio(user_balance, last_debt_token_value, token_client.total_supply)
```

Snippet 4.13: Code snippet from `get_info_by_address`.

The issue lies in the use of `token_client.total_supply` instead of `SCALE_VALUE` for division. The `last_debt_token_value` is denominated in `stable_coin / security_token` units and does not represent the total amount of `stable_coin` tokens.

Impact The function `get_info_by_address` will provide inaccurate information about the user's holdings, potentially leading to erroneous financial decisions by 3rd party integrators based on incorrect data.

Recommendation Change `token_client.total_supply` for `SCALE_VALUE`.

Developer Response

4.1.16 V-HYD-VUL-016: Malicious user can manipulate the initial exchange rate of the passive pool

Severity	Low	**Commit**	1e0f9af
Type	Data Validation	**Status**	Acknowledged
File(s)	`passive_pool/contract.rs`		
Location(s)	_redeem		
Confirmed Fix At	N/A		

The `passive_pool` contract initializes with a default exchange rate of `1e7`. Over time, this rate is expected to increase as interest accrues. However, a vulnerability exists where a malicious user can manipulate the exchange rate to appear 1000 times higher than the initial rate, potentially impacting third-party contracts. Here's how the manipulation works:

1. A user deposits `1` stablecoin, receiving `1e3` share tokens based on the `supply` calculation.
2. The user then redeems `1e3 - 1` shares. The math in `_redeem` calculates:
   - `redeem_amount = (1e3 - 1) * 1e7 / 1e10`.
   - This results in zero due to integer division.
3. After these operations, the contract's state shows:
   - Total stablecoin = 1
   - Total supply of shares = 1

Subsequently, when the `rate` function is called, it returns `1e10`, misleadingly suggesting a 1000-times increase in the exchange rate.

Impact The misleadingly high exchange rate reported by the `rate` function can cause issues with third-party protocols that rely on this metric without validating the real stable coin balance and shares reserves, leading to incorrect operational decisions.

Recommendation The contract should revert when the `redeem_amount` computes to zero.

Developer Response

4.1.17 V-HYD-VUL-017: Linked_list operations are not checked for success

Severity	Low	**Commit**	1e0f9af
Type	Data Validation	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	_queue_process, _cancel_withdraw, _withdraw		
Confirmed Fix At	N/A		

The `credit_vault` contract uses the `push_back` and `remove` to build the withdrawal queue linked list. Currently, the `credit_vault` contract does not validate return values of these functions: `push_back` that returns a boolean indicating success, and `remove` that returns `0` if it fails.

Impact Failing to validate these return values means that errors during these operations are not detected, which could lead to silent failures and continued execution. This oversight may result in inconsistent states within the withdrawal queue.

Recommendation Implement checks to validate the return values from `push_back` and `remove` .

Developer Response Recommendation implemented.

4.1.18 V-HYD-VUL-018: Credit Vault cannot make admin actions over security token

Severity	Low	**Commit**	1e0f9af
Type	Maintainability	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	See description		
Confirmed Fix At	N/A		

The `credit_vault` contract serves as the administrator for the deployed `security_token`. Within the `security_token` contract, the administrator has the authority to upgrade the contract, assign a new admin, and set a different compliance contract. Despite this administrative capability, the current `credit_vault` contract lacks direct functionality to invoke these admin-restricted functions on the `security_token`.

Impact The absence of this functionality in the `credit_vault` means that any necessary administrative actions on the `security_token` must be managed through an upgrade of the `credit_vault` itself. This approach is not only cumbersome but also prone to errors and delays, potentially affecting the security of the contract.

Recommendation It is recommended to extend the `credit_vault` contract with dedicated functions that enable it to directly administer the `security_token`.

Developer Response Recommendation was implemented.

4.1.19 V-HYD-VUL-019: Role should default to admin role during initialize

Severity	Low	**Commit**	1e0f9af
Type	Maintainability	**Status**	Acknowledged
File(s)	`access_control/lib.rs`		
Location(s)	initialize		
Confirmed Fix At	N/A		

The `initialize` function in the `AccessControl` struct implementation currently accepts a `role` parameter. However, the intended design requires that the `DEFAULT_ADMIN_ROLE` be granted during initialization for the system to function correctly. Allowing the caller to specify a different role can lead to misconfiguration.

Impact Allowing a role other than the `DEFAULT_ADMIN_ROLE` to be specified during the initialization will break the access control system.

Recommendation It is recommended to eliminate the `role` parameter from the `initialize` function and directly use the `DEFAULT_ADMIN_ROLE`.

Developer Response

4.1.20 V-HYD-VUL-020: Utilization rate will be computed incorrectly due to arguments being flipped

Severity	Low	**Commit**	1e0f9af
Type	Logic Error	**Status**	Acknowledged
File(s)	`isolated_pool/contract.rs`		
Location(s)	supply_rate		
Confirmed Fix At	N/A		

In the `supply_rate` function of the `isolated_pool` contract, there is an incorrect ordering of parameters when calling the `utilization_rate` function. The `supply_rate` function passes `cash` as the first parameter and `borrows` as the second, which is the opposite of the expected order in `utilization_rate`, where `borrows` should be first and `cash` second. This is illustrated in the following code snippet:

```
fn supply_rate(e: &Env, cash: i128, borrows: i128, reserves: i128, reserve_factor:
    i128) -> i128 {
    // ....
    // cash and borrows are flipped.
    utilization_rate(cash, borrows, reserves) * rate_to_pool / SCALE_VALUE
}

fn utilization_rate(borrows: i128, cash: i128, reserves: i128) -> i128 {
    // ....
}
```

Snippet 4.14: Code snippet from the `isolated_pool`.

Impact This parameter misordering leads to incorrect computation of the supply rate, potentially affecting interest calculations and interactions with third-party contracts or systems that monitor or depend on these values for financial decisions or analytics.

Recommendation Correct the argument order in the `supply_rate` function call to `utilization_rate` by passing `borrows` as the first argument and `cash` as the second.

Developer Response

4.1.21 V-HYD-VUL-021: Input sanitization best practices

Severity	Low	**Commit**	1e0f9af
Type	Data Validation	**Status**	Fixed
File(s)	`token/contract.rs`, `security_token/contract.rs`, `credit_vault/contract.rs`		
Location(s)	initialize		
Confirmed Fix At	`https://github.com/lydialabs/HiYield-stellar-contracts/pull/48`		

Making explicit the intended range of values through input sanitization checks is crucial for maintaining the integrity of contract operations. These checks can prevent potential unintended behaviors that may arise from user errors, administrative mistakes, or future integrations.

`credit_vault/contract.rs`:

- The following values provided in `init_data` should be validated to be greater than zero: `init_data.total_assets`, `init_data.last_time_accrued` and `init_data.total_borrowed_assets`.
- The `security_token` name and symbol should be checked to not be empty strings during the `initialize` function.

`isolated_pool/contract.rs`:

- The value that is written as the `BaseRatePerSecond` during `initialize` should be checked to not be greater than `BORROW_RATE_MAX` since the `accrue_interest` function will panic for higher values.

Impact Although the current contracts are not directly vulnerable to exploits due to the lack of the previous recommended checks, implementing strict validation of input values is a prudent measure to increase robustness and prevent misuse in different implementations context.

Recommendation See description.

Developer Response Recommendation implemented.

4.1.22 V-HYD-VUL-022: Centralization Risk

Severity	Low	**Commit**	1e0f9af
Type	Centralization Risk	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

The upgradeability of smart contracts in this system, particularly in the `ai_pool`, `isolated_pool`, `passive_pool`, and `credit_vault`, introduces a centralization risk by granting the admin the capability to execute arbitrary logic. This centralized control places significant power in the hands of a single entity, which is also responsible for fulfilling crucial financial obligations such as debt repayment and interest.

Impact The ability of the admin to execute arbitrary changes to the code and control financial operations introduces a significant risk of misuse or mismanagement. It places the entire system's stability and integrity at the mercy of the admin's actions. In the financial context of `ai_pool`, `isolated_pool`, `passive_pool` and `credit_vault` the necessity for the admin to meet debt and interest obligations further heightens the risk, as any failure to perform these duties could lead to financial instability and loss of user trust.

Recommendation It is advisable to clearly document the trust assumptions for users who utilize these contracts. For instance, users should be informed that the admin has the ability to change the code of the contract and in the case of the `credit_vault` and `ai_pool` products the admin can take uncollateralized loans from depositors.

Developer Response

4.1.23 V-HYD-VUL-023: Insufficient number of parameters supplied to token_client initialize call

Severity	Low	**Commit**	1e0f9af
Type	Invalid Interface	**Status**	Fixed
File(s)	`ai_pool/contract.rs`		
Location(s)	initialize		
Confirmed Fix At	N/A		

The `initialize()` function of the `ai_pool` contract incorrectly calls the `initialize()` function of a token with an insufficient number of arguments. The current implementation attempts to initialize the `token_client`, which interfaces with the `security_token` contract, using the following parameters:

```
token_client.initialize(
        &e.current_contract_address(),
    &10u32,
    &token_name,
    &token_symbol,
);
```

Snippet 4.15: Code snippet from `initialize()`.

However, the `security_token` contract's `initialize()` function requires an additional parameter, as shown in its interface definition:

```
pub fn initialize(e: Env, admin: Address, operator: Address, decimal: u32, name:
    String, symbol: String)
```

Snippet 4.16: `initialize` interface definition.

Impact The discrepancy in the number of parameters leads to a compilation failure. The project currently compiles only because outdated `.optimized.wasm` files are present in the `target/wasm32-unknown-unknown/release/` directory, masking the issue.

The root cause of the issue is explained in detail in a previous entry: V-HYD-VUL-005.

Recommendation To correct this issue, the call to `token_client.initialize()` within the `ai_pool` contract's `initialize()` function should be updated to include all required parameters as specified in the latest `security_token` contract. Additionally, the build process should be reviewed and fixed to ensure that stale `.wasm` files do not remain in the repository, preventing similar issues from going unnoticed in the future.

Developer Response Recommendation implemented.

4.1.24 V-HYD-VUL-024: Insufficient input validation in claim_rewards and transfer_rewards

Severity	Low	**Commit**	1e0f9af
Type	Logic Error	**Status**	Acknowledged
File(s)	`rewards/contract.rs`		
Location(s)	claim_rewards, transfer_rewards		
Confirmed Fix At	N/A		

In the functions `claim_rewards` and `transfer_rewards` of the `rewards` contract, the input validation checks for `amount` and `supply` parameters are insufficient. It allows both values to be negative.

```
if amount > supply {
    panic!("Reward: Amount param must be less or equal to supply param");
}
```

Snippet 4.17: Code snippet from `claim_rewards`.

Impact If the `amount` and `supply` are negative, the recipient of the reward will be able to receive more reward tokens than they would normally. The current `ai_pool` contract integrates correctly with the rewards contract, but if this code will be used with other pools, this issue needs to be addressed.

Recommendation Panic if both the `amount` and `supply` are negative.

Developer Response

4.1.25 V-HYD-VUL-025: Scale value is not tied to token decimals

Severity	Warning	**Commit**	1e0f9af
Type	Maintainability	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

In the contracts for `credit_vault`, `ai_pool`, `passive_pool`, and `isolated_pool`, the initialization of the `share` token includes setting a hard-coded number of decimal places that the token supports. Additionally, the `SCALE_VALUE` parameter used throughout the code to rescale token values to `share` decimals is derived from this hardcoded value, thus tying it closely to the shares token's decimal configuration.

Impact This setup presents a maintainability challenge. If there is ever a need to adjust the `SCALE_VALUE` or change the token's decimal precision, these hard-coded values could lead to inconsistencies and errors in the contract's functionality. The interdependence of these values means that an update to one necessitates a corresponding update to the others, increasing the potential for oversight and error during maintenance or upgrades.

Recommendation To improve system maintainability and ensure consistency, it is advisable to programmatically link the `SCALE_VALUE` with the decimals set to the `share` token. For example, it is recommended to define a constant for the token's decimal precision, such as `SHARE_TOKEN_PRECISION` and computing as `SCALE_VALUE = 10^SHARE_TOKEN_PRECISION`.

Developer Response

4.1.26 V-HYD-VUL-026: Hard assumption around token decimals

Severity	Warning	**Commit**	1e0f9af
Type	Usability Issue	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	initialize		
Confirmed Fix At	N/A		

During the `initialize` function, the code calculates and stores a decimal offset based on the assumption that the asset token has 7 or fewer decimals. This assumption conflicts with the more common "consensus" across various ecosystems, where token decimals are typically set to 18.

Impact The assumption that asset tokens will have no more than 7 decimals means that the contract will not be compatible with tokens that use the standard 18 decimals.

Recommendation To address this issue, the contract should be modified to support tokens with up to 18 decimals, which is the standard used by most tokens. Alternatively, if not, it is important to clearly document the contract's limitation regarding tokens' decimal support.

Developer Response Recommendation was implemented.

4.1.27 V-HYD-VUL-027: Inconsistency on interest computation

Severity	Warning	**Commit**	1e0f9af
Type	Data Validation	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	total_assets		
Confirmed Fix At	N/A		

The `_total_assets` function calculates its return value by summing `read_total_assets` and `interest_assets`. This calculated value is critical as it is used to determine the quantity of shares or assets for operations such as minting, redeeming, depositing, or withdrawing. However, there is an inconsistency in how `total_assets` is updated during the execution of `accrued_interest`. Specifically, the `accrued_interest` function only updates `total_assets` when the computed interest is positive.

Impact if the computed interest is negative, the contract will mistakenly overestimate the number of shares to be minted for a depositor or underestimate the number of assets required for a deposit.

Recommendation To mitigate this issue, it is advisable to adjust the `_total_assets` function so that it returns only `read_total_assets` when the computed interest is negative.

Developer Response Recommendation was implemented.

4.1.28 V-HYD-VUL-028: Inconsistency on the check for the max_utilization parameter

Severity	Warning	**Commit**	1e0f9af
Type	Data Validation	**Status**	Acknowledged
File(s)	`passive_pool/contract.rs`		
Location(s)	update_max_utilization		
Confirmed Fix At	N/A		

The `initialize()` function of the contract has an inconsistency in its validation logic for `pool_init.max_utilization` compared to the validation in the `update_max_utilization` function. Specifically, the code does not allow `pool_init.max_utilization` to be equal to the `UPPER_BOUND` . In contrast, the `update_max_utilization` function permits `max_utilization` to be equal to `UPPER_BOUND`.

Impact This inconsistency can lead to confusion and potentially erroneous behavior, as the permissible range for `max_utilization` changes depending on whether it is being initialized or updated.

Recommendation To ensure consistency and prevent configuration errors, align the checks across both functions. A recommended approach is to allow `max_utilization` to be equal to `UPPER_BOUND` in `initialize`.

Developer Response

4.1.29 V-HYD-VUL-029: pool_info is not implementing the interface correctly

Severity	Warning	**Commit**	1e0f9af
Type	Maintainability	**Status**	Acknowledged
File(s)	`ai_pool/contract.rs`		
Location(s)	pool_info		
Confirmed Fix At	N/A		

The `ai_pool` contract's interface specification for the `pool_info` function outlines a specific order of returned values, which is not adhered to in the actual implementation. The mismatch in the ordering of output values can lead to confusion and potential misuse of the function. Specified order:

1. Total underlying token
2. Available balance on the pool
3. Total borrowed
4. Last timestamp accrued
5. Pool ratio

In the below implementation, the order of returned values does not match the interface description, and notably, a `0` is returned for the "total borrowed" value without clear documentation or explanation.

```
fn pool_info(e: Env) -> (i128, i128, i128, i128, i128) {
        let (last_debt_token_value, _) = _debt_token_value_lastest(&e);
        (
            get_ratio(
                security_token::Client::new(&e, &read_token_contract(&e)).
    total_supply(),
                last_debt_token_value,
                SCALE_VALUE
            ),
            token::Client::new(&e, &read_stable_coin(&e)).balance(&e.
    current_contract_address()),
            0,
            read_accrual_timestamp(&e),
            last_debt_token_value,
        )
 }
```

Snippet 4.18: Code snippet from `ai_pool`. Implementation of the `pool_info` function.

Impact The discrepancy between the interface documentation and the actual implementation can lead to incorrect assumptions and decisions based on the data returned by `pool_info`. Users and third-party integrations relying on the specified order might misinterpret the data, leading to potential errors in their operations or analyses.

Recommendation To address the inconsistencies and improve clarity in the `ai_pool` contract, it is recommended to realign the `pool_info` function's implementation to match the order specified in the interface documentation.

Developer Response

4.1.30 V-HYD-VUL-030: Issues when using declared interfaces

Severity	Warning	**Commit**	1e0f9af
Type	Maintainability	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

The usage of interfaces across the `isolated_pool` and `passive_pool` contracts demonstrates inconsistencies and potential confusion in how these interfaces are implemented and named:

1. **Incorrect Arguments in Contract Creation**: In `isolated_pool` and `passive_pool`, the `create_contract` function expects a `stable_coin` address as the second argument. However, the `passive_pool` and `pool_owner` addresses are mistakenly passed instead.
2. **Interface Naming Inconsistencies**: In the `isolated_pool`, there are misalignments in the type names used for similar interface functions. For example, the `token2::Client` is ambiguously used for both underlying token interactions and share token interactions, which can lead to confusion:

```
fn get_cash_prior(e: &Env) -> i128 {
    let cash = token2::Client::new(&e, &_get_underlying_token(e)).balance(&e.
    current_contract_address()) - read_type_i128(e, ISDataKey::TotalCanClaim);
    ...
}

fn exchange_rate_stored_internal_(e: &Env, total_cash: i128, total_borrows: i128,
    total_reserves: i128) -> i128 {
    let total_supply = token2::Client::new(&e, &read_type_address(e, ISDataKey::
    TokenAddress)).total_supply() + read_type_i128(e, ISDataKey::PassivePoolBalance);
    ...
}
```

Snippet 4.19: Code snippet from the `contract` file in the `isolated_pool` crate.

Impact While these issues do not directly lead to exploits, they complicate maintenance and can be sources of confusion.

Recommendation It is advisable to be consistent with interfaces types and definitions.

Developer Response

4.1.31 V-HYD-VUL-031: Missing event on state update

Severity	Warning	**Commit**	1e0f9af
Type	Missing/Incorrect Eve	**Status**	Acknowledged
File(s)	`passive_pool/contract.rs`		
Location(s)			
Confirmed Fix At			

Upon a state update, it is strongly recommended that developers emit an event to indicate that a change has been made. This allows both external users and protocol administrators to monitor the protocol for various reasons, including potentially suspicious activity. Therefore, it is critical for significant changes to the protocol to be accompanied by events in order to enable this monitoring.

The `borrow` function of the `passive_pool` contract, however, does not emit any event when the underlying assets get transferred from the `passive_pool` contract address to the `isolated_pool` contract address. For some reason, the event is commented out.

```
// emit event
// event::loan(&e, from, borrow_amount);
```

Snippet 4.20: A snippet from the `credit_vault::borrow` function

Impact The borrow action executed on the passive pool will stay unnoticed. External users and protocol administrators may miss a potentially suspicious activity.

Recommendation It is recommended to bring back the `loan` event.

4.1.32 V-HYD-VUL-032: AccessControl implementation lacks function to change the role admin

Severity	Warning	**Commit**	1e0f9af
Type	Usability Issue	**Status**	Acknowledged
File(s)	`access_control/lib.rs`		
Location(s)	See description		
Confirmed Fix At	N/A		

The `access_control` crate is designed to facilitate a flexible system of access control via a roles system, allowing specific roles to administer others. The `write_role_admin` function is crucial for setting up these administrative roles. However, the current implementation of the `AccessControl` struct lacks a public function to access `write_role_admin`.

Impact Without public access to the `write_role_admin` function, users of the `AccessControl` struct are unable to configure or modify the roles that have the authority to grant or revoke other roles. This limitation restricts the functionality of the access control system.

Recommendation It is recommended to introduce a public function that provides access to the `write_role_admin` function.

Developer Response

4.1.33 V-HYD-VUL-033: Access control roles should be saved into persistent storage

Severity	Warning	**Commit**	1e0f9af
Type	Denial of Service	**Status**	Acknowledged
File(s)	`access_control/lib.rs`		
Location(s)	write_role		
Confirmed Fix At	N/A		

The `access_control` crate is designed to facilitate a flexible system of access control via a roles system, which allows for the creation of specific roles to administer others. Currently, the code stores the role keys and admin role keys in the instance storage rather than in persistent storage.

Impact Using instance storage, which has limited capacity, restricts the scalability of the access control system, preventing the creation of more complex role hierarchies. Additionally, because instance storage is tied to the contract instance, it increases the cost for users interacting with the contract due to the need to load the entire access control system for each contract invocation.

Recommendation To enhance scalability and reduce operational costs, it is advisable to store the role keys and admin role keys in persistent storage instead of instance storage.

Developer Response

4.1.34 V-HYD-VUL-034: Duplicated code

Severity	Info	**Commit**	1e0f9af
Type	Maintainability	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	initialize		
Confirmed Fix At	N/A		

The `initialize` function in the contract unnecessarily writes the administrator information to the storage two times: first, it calls a dedicated function called `write_administrator` and then it directly calls the `set` function from the SDK to write to the instance storage.

Impact The duplication of code, especially code that interacts with storage, can make maintainability more difficult and incur additional costs for end users.

Recommendation It is advisable to remove the duplicated code.

Developer Response Recommendation implemented.

4.1.35 V-HYD-VUL-035: Pending withdraw assets variable lacks external getter

Severity	Info	**Commit**	1e0f9af
Type	Maintainability	**Status**	Fixed
File(s)	`credit_vault/contract.rs`		
Location(s)	pending_withdraw_assets		
Confirmed Fix At	N/A		

The variable `pending_withdraw_assets` tracks the amount of assets that have been withdrawn but cannot be claimed yet. The contract does not include an external getter to read the value of this variable, making it much harder to query this information off-chain.

Impact The lack of an external getter for `pending_withdraw_assets` reduce the ability to monitor these values from outside the contract: for example off-chain monitors or external contracts.

Recommendation It is recommended to add an external getter function for the `pending_withdraw_assets` variable.

Developer Response Recommendation was implemented.

4.1.36 V-HYD-VUL-036: Code repetition in several places

Severity	Info	**Commit**	1e0f9af
Type	Maintainability	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

Several smart contracts within the project use an `admin`module that provides access control functions, such as `only_admin()`. Although these functions are available, the code in multiple contracts often replicates the access control checks directly instead of using the standardized functions provided by the `admin` module.

```
pub fn burn_by_admin(e: Env, from: Address, amount: i128) {
    check_nonnegative_amount(amount);
    let admin = read_administrator(&e);
    admin.require_auth();
    ...
}
```

Snippet 4.21: Snippet from `token/contract.rs` file:

This pattern is repeated across various contracts, including:

- `ai_pool/contract.rs`
- `passive_pool/contract.rs`
- `security_token/contract.rs`
- `standalone_ai_pool/contract.rs`
- `complience/contract.rs`
- `credit_vault/contract.rs`
- `rewards/contract.rs`
- `token/contract.rs`

Impact Replicating access control checks instead of using a specially designated function can significantly hinder the maintainability of the project. It increases the risk of errors during updates and makes managing the code-base more challenging.

Recommendation To enhance maintainability and consistency across the project, it is recommended to utilize the standard access control functions provided in the `admin` module, such as `only_admin()`, instead of duplicating the checks.

Developer Response

4.1.37 V-HYD-VUL-037: Documentation Improvements

Severity	Info	**Commit**	1e0f9af
Type	Maintainability	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

On the documentation website, for the AI pool, there is a following description:

```
The amount of rewardTokens received is calculated as follows:

rewardTokens = redeemAmt / 1,000,000 * rewardTokensAvailable
```

Snippet 4.22: Text from the project's documentation.

However, the implementation calculates the reward tokens differently. It is calculated as follows:

```
rewardTokens = redeemAmt / debtTokensSupply * rewardTokensAvailable
```

Impact Inconsistency between the docs and the implementation may affect maintainability and usability of the system.

Recommendation It is recommended to make the docs consistent with the implemented code.

4.1.38 V-HYD-VUL-038: The linked list size should not be allowed to change externally

Severity	Info	**Commit**	1e0f9af
Type	Maintainability	**Status**	Fixed
File(s)	`credit_vault/linked_list.rs`		
Location(s)	write_size		
Confirmed Fix At	N/A		

In the `linked_list` implementation, the `write_size` function is declared as public, which allows developers to alter the size of the linked list directly.

Impact By making the `write_size` function public, developers can modify the size of the linked list independently of the standard operations that add or remove its elements. This opens up the possibility for discrepancies between the actual number of elements and the size value, leading to potential integrity issues within the linked list's structure.

Recommendation It is advisable to not declare the `write_size` function as public. Ensuring that the size of the linked list can only be modified through controlled operations (such as adding or removing elements) will maintain the integrity and consistency of the linked list data structure.

Developer Response Recommendation was implemented.