certora

Security Assessment & Formal Verification Report

Huma protocol

July 2024

Prepared for 00labs

Table of content

certora

Project Summary **4**
Project Scope 4
Project Overview 4
Findings Summary 4
Severity Matrix 5
Detailed Findings **6**
Low Severity Issues 6
L-01 Initialize can be front-run 6
L-02 Round down in cancel_redemption_request() 7
L-03 Round down in withdraw_after_pool_closure() 9
L-04 Pool state can be changed after it is closed 9
L-05 Underlying token decimals could be too large or too small 11
L-06 Removing and adding lender deletes data 12
L-07 Unprocessed_amount not always scaled correctly 13
L-08 Event uses amount instead of amount_to_collect 15
L-09 Misleading events 15
L-10 Duplicate error codes 16
L-11 Missing check on index 18
Informational Severity Issues 18
I-01. Enum element ProtocolOn not used 18
I-02. Comment make_principal_payment() incorrect 19
I-03. Comment process_redemption_requests() outdated 19
I-04. Comment get_days_diff() outdated 19
I-05. Function withdrawable_assets() could use lrr.withdrawable() 20
I-06. Function drawdown() could return data 21
I-07. Function get_amount_available_for_drawdown() could check more 21
I-08. Steller/Rust and Solidity code differs slightly 21
I-09. Function make_principal_payment() could use <= 22
I-10. Inconsistent error codes 23
I-11. Newer version of library chrono is available 23
I-12. Rust function to calculate the number of days could be used 24
I-13. Clearer comments 25

I-14. Confusing name for State 27
Appendix A **28**
Appendix B **29**
Disclaimer **30**
About Certora **30**

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Huma protocol	https://github.com/00labs/huma-soroban-contracts	Original: e1ceb0f	Stellar
Huma protocol	https://github.com/00labs/huma-soroban-contracts	Audit fixes: ba14880	Stellar
Huma protocol	https://github.com/00labs/huma-soroban-contracts	Further changes: 89d1b6 (see Appendix A below)	Stellar

Project Overview

The Huma protocol is a lending protocol that provides on-chain private credit facilities for businesses.

The Huma protocol is available for multiple blockchain platforms. This review focuses on the version for Stellar / Rust.

We have verified the fixes that are present in this commit hash: ba14880.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-

High	-	-	-
Medium	-	-	-
Low	11	11	9
Informational	14	14	10
Total	25	25	19

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

Low Severity Issues

L-01 Initialize can be front-run		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: pool_manager - contract.rs pool_storage - contract.rs pool - contract.rs tranche_vault - contract.rs huma_config - contract.rs credit - contract.rs credit_manager - contract.rs credit_storage - contract.rs	Status: Fixed	

Description: Most crates contain a `contract.rs` with the function `initialize()`. The functions are not authorized, but they can only be run once.

```
JavaScript
pub fn initialize(...) {
    // Only allow initialization once
    ...
}
```

Exploit Scenario: During deployment or an `upgrade()`, these functions could be called by an unauthorized user, who is monitoring the deployment of contracts via the mempool. These could lead to contracts that are not correctly configured and possibly taken over.
During initial deployment this could be detected and the contract could be redeployed.

During an upgrade this risk might be higher, depending on the fact if there would be an additional `initialize()` function to initialize the update. If that would be the case then an active contract might be unrecoverable.

Recommendations: Consider using authorization in the `initialize()` function. Possibly use precomputed deterministic deployed contract addresses.

Customer's response: We check the deployment and deploy and initialize another instance if necessary. It is unlikely that an upgrade requires an additional initialization.
We will discuss with the team what fix we want to proceed with.

After discussing with the team, we've decided to add a contract factory and bundle deployment and initialization together.

L-02 Round down in cancel_redemption_request()		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
File: tranche_vault.rs	Status: Confirmed, will not be fixed	

Description: In function cancel_redemption_request() , the following statement rounds down the principal that is recovered:

```
JavaScript
dr.cancel_redemption(lrr.principal_requested() * shares / lrr.num_shares_requested());
```

In extreme cases, for example when `lrr.principal_requested() < lrr.num_shares_requested()` and `shares_cancelled == 1`, no principle is recovered.

Exploit Scenario:

```javascript
let principal_deposited: u128 = 1;
let total_shares: u128 = 2;
let mut lrr:LenderRedemptionRecord = LenderRedemptionRecord::new_empty();
let mut dr:DepositRecord = DepositRecord::new_empty();
let mut ers:EpochRedemptionSummary = EpochRedemptionSummary::new_empty();
dr.deposit(principal_deposited,1);

let shares_requested: u128 = 2; // so 100% of principal
let principal_requested = (dr.principal() * shares_requested) / total_shares; // 1
ers.add_redemption(shares_requested);
lrr.add_redemption(shares_requested, principal_requested);
dr.add_redemption(principal_requested);

let shares_cancelled: u128 = 1;
for _n in 1..=2 {
    if lrr.num_shares_requested() < shares_cancelled {
        panic_with_error!(env, TrancheVaultError::InsufficientSharesForRequest);
    }
    let tmp = lrr.principal_requested() * shares_cancelled / lrr.num_shares_requested(); // 1
* 1 / 2 = 0
    dr.cancel_redemption(tmp); // self.principal += assets;
    lrr.cancel_redemption(shares_cancelled);
    ers.cancel_redemption(shares_cancelled);
}
return dr.principal(); // is now 0
```

Recommendations: Consider detecting when the value rounds down to 0 and then revert the call.

Customer's response: Since this number is only used for book-keeping purposes, the lender has nothing to gain by performing the attack above. In addition, since we have a minimum deposit requirement, the lender has to repeat the loop above for 10^8 (exponent varies depending on the decimals of the underlying token) times in order to get the desired result.

L-03 Round down in withdraw_after_pool_closure()		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
File: tranche_vault.rs	Status: Confirmed, will not be fixed	

Description: Function `withdraw_after_pool_closure()` uses `convert_to_assets()` which rounds down. So in the end there might be some asset tokens left after all `withdraw_after_pool_closure()` have been done.

Exploit Scenario:

Recommendations: Consider having a way to recover the funds or alternatively leave them when they are negligible.

Customer's response: This is true. The amount left should be negligible though (< $1 collectively) so this shouldn't be a concern. I don't think there is a good way to avoid this anyway due to the presence of rounding. In addition, one of our principles is to limit admins' power, so we don't think it's a good practice to create a special channel for the admins to withdraw the residual money in this case.

L-04 Pool state can be changed after it is closed		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
File: pool_manager.rs	Status: Fixed	

Description: The functions `enable_pool()`, `disable_pool()` and `close_pool()` don't check for the previous state. This allows a closed pool to be reopened again.
Reopening a closed pool can lead to unexpected situations.
According to the Huma project, there might be circumstances where reopening a closed pool is necessary, which deviates from the comment at the `enum PoolStatus`.

```
JavaScript
pub enum PoolStatus {
    Off,    // The pool is temporarily turned off.
    On,     // The pool is active.
    Closed, // The pool is permanently closed after maturity.
}

pub(crate) fn enable_pool(env: &Env, pool_storage: &PoolStorageClient, caller: &Address) {
    ... // no checks on pool status
    pool_storage.set_pool_status(&PoolStatus::On);
    ...
}

pub(crate) fn disable_pool(env: &Env, pool_storage: &PoolStorageClient, caller: &Address) {
    ... // no checks on pool status
    pool_storage.set_pool_status(&PoolStatus::Off);
    ...
}

pub(crate) fn close_pool(env: &Env, pool_storage: &PoolStorageClient, caller: &Address) {
    ... // no checks on pool status
    pool_storage.set_pool_status(&PoolStatus::Closed);
    ...
}
```

Exploit Scenario: Although access to the functions is authorized, mistakes can be made.

Recommendations: Double-check which transitions are allowed. Carefully check the side effects of the transition from `Closed` to `Open` if it would be allowed. Check the previous states and disallow unwanted transitions. Make sure the comments (of PoolStatus - `The pool is permanently closed after maturity`) are in line with the allowed state changes.

Customer's response: We can make the following fixes:

- Do not allow the same operation to be carried out twice (e.g. double enabling)
- Do not allow a pool to be disabled after it's closed.

L-05 Underlying token decimals could be too large or too small		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: pool_manager.rs due_manager.rs	Status: Fixed	

Description: The number of decimals of the underlying token is not checked. If they would be too large then this statement would fail: `10_u128.pow(token.decimals())`.

Also, calculations with `DEFAULT_DECIMALS_FACTOR` might have insufficient precision.

JavaScript

```
pub(crate) fn set_pool_settings(...) {
    let token = TokenClient::new(env, &pool_storage.get_underlying_token());
    if min_deposit_amount < MIN_DEPOSIT_AMOUNT_THRESHOLD * 10_u128.pow(token.decimals()) {
        ...
    }
    ...
}
```

JavaScript

```
pub const DEFAULT_DECIMALS_FACTOR: u128 = 1e18 as u128;
```

Additionally in function `refresh_late_fee()` values can be rounded down to `0` if too few decimals are used. If the number of decimals is `2`; and the amount is low: for example 10.00; with 4 days and fee_bps of 500, the fee could be rounded down to `0`:
`500 * 1000 * 4 / (10000 * 360)` ==> `0,56` ==> rounded down to `0`.

```
JavaScript
(late_fee_basis
* (late_fee_bps as u128)
* calendar::get_days_diff(late_fee_start_date, late_fee_updated_date).unwrap_optimized()
as u128)
/ (HUNDRED_PERCENT_BPS * DAYS_IN_A_YEAR) as u128
```

Also, see Soroban examples PR 309.

Exploit Scenario: An underlying token with a small or large number of decimals could be used. However, this is unlikely because the configuration is authorized, and normally stablecoins are used, which are unlikely to have a large number of decimals.

Recommendations: Consider restricting the number of decimals to a reasonable amount, for example, at least `6` and at the most `24`.

Customer's response: The chance of us using stable coins with too few or too many decimals is very small. We can add a check to make sure the token has 6-18 decimals (18 coming from the example PR linked in the issue description). / ok cool. We can update `tranche_token` to do the same 18-decimal check as in the Soroban examples.

L-06 Removing and adding lender deletes data		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: tranche_vault.rs	Status: Fixed	

Description: When doing `remove_approved_lender()` and then `add_approved_lender()`, the `DepositRecord` is overwritten. However the `shares` are still present so the funds can still be retrieved.
The `last_deposit_time` will also be `0` and `add_redemption_request()` can be done immediately, even if the `withdrawallockout` period hasn't passed compared to the original `deposit` time. In practice this seems unlikely to be an issue.

According to the Huma project the removing of a lender can happen if a lender is detected to be a sanctioned party and needs to be prevented from supplying additional capital. In extremely rare cases the lender could be re-added.

Exploit Scenario: Call `remove_approved_lender()` and then `add_approved_lender()`.

Recommendations: Consider in `add_approved_lender()` to keep the `DepositRecord`.

Customer's response: What we might be able to do is to detect whether there is an existing `DepositRecord` for the lender and re-use the existing one when the lender is added again.

After discussing with the team, we will require the lender to withdraw all their assets before being eligible to be re-added as a lender. In case a lender is removed due to regulatory actions, the regulatory authority may withdraw the assets of the lender by gaining control of their wallet first.

To that end, we will add a check in `add_approved_lender()` to prevent an account from being approved if the lender has remaining assets. If not, then we will allow the lender to be re-added.

L-07 Unprocessed_amount not always scaled correctly		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
File: epoch_manager.rs	Status: Fixed	

Description: The first calculation of `unprocessed_amount` in process_epoch() does not divide by `DEFAULT_DECIMALS_FACTOR`, but the second calculation does. Because the second calculation isn't always executed, the `unprocessed_amount` in the event could be scaled incorrectly.

JavaScript

```
process_epoch(...) {
    ...
    let mut unprocessed_amount = (junior_summary.total_shares_requested()
        - junior_summary.total_shares_processed())
        * junior_price; // not scaled
    ...
    if let Some(senior_addr) = senior_addr_opt {
        ...
        unprocessed_amount = (unprocessed_amount
            + ((senior_summary.total_shares_requested()
                - senior_summary.total_shares_processed())
                * senior_price))
            / DEFAULT_DECIMALS_FACTOR;
        ...
    }
    ...
    env.events().publish(
        (Symbol::new(env, "RedemptionRequestsProcessed"),),
        RedemptionRequestsProcessedEvent {
            ...
            unprocessed_amount, // sometimes scaled, sometimes not
        },
    );
}
```

Exploit Scenario: Call the function `process_epoch()` without a senior tranche.

Recommendations: In both calculations, divide by `DEFAULT_DECIMALS_FACTOR`.

Customer's response: You are right. We will divide by `DEFAULT_DECIMALS_FACTOR` in both cases. The impact is small though since it's only used in the event, not in the actual calculation.

L-08 Event uses `amount` instead of `amount_to_collect`		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
File: credit.rs	Status: Fixed	

Description: The function make_principal_payment() publishes an event with `amount` while the comparable function `make_payment()` uses `amount_to_collect`. Also the equivalent function in Solidity uses `amountToCollect`.

```
JavaScript
env.events().publish(
    (Symbol::new(env, "PrincipalPaymentMade"),),
    PrincipalPaymentMadeEvent {
        ...
        amount, // should be amount_to_collect
        ...
    },
);
```

Exploit Scenario: Call function `make_principal_payment()` and observe the events.

Recommendations: Replace `amount` with `amount_to_collect`.

Customer's response: We will update this.

L-09 Misleading events		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**

Files: huma_config.rs pool_manager - contract.rs tranche_vault.rs	Status: Fixed	

Description: Several functions publish events while that might not be relevant because the system hasn't changed state. These events might be misinterpreted.
We have found the following functions:

- set_liquidity_asset()
- add_pauser() / remove_pauser()
- pause_protocol() / unpause_protocol()
- remove_pool_operator()
- remove_approved_lender()

Exploit Scenario: Calling any of the functions twice.

Recommendations: Consider detecting if the system is already in the desired state and then skip publishing the event.

Customer's response: Since it's just an event we'll treat this as low priority.

L-10 Duplicate error codes		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: huma_config - errors.rs calendar.rs tranche_vault - errors.rs pool_manager - errors.rs	Status: Fixed	

Description: The error code `101` is defined twice, this might be confusing with troubleshooting.

```
JavaScript
pub enum HumaConfigError {
    PauserRequired = 101,
    ...
}
```

```
JavaScript
pub enum CalendarError {
    StartDateLaterThanEndDate = 101,
}
```

Additionally the error code `PoolOperatorRequired` is defined twice, with error 301 & 401, however this is less of an issue because there is no risk of confusion.

```
JavaScript
pub enum TrancheVaultError {
    PoolOperatorRequired = 401,
    ...
}
```

```
JavaScript
pub enum PoolManagerError {
    PoolOperatorRequired = 301,
    ...
}
```

Exploit Scenario: Call two different functions that result in the two different errors.

Recommendations: Consider changing one of the `101` errors to a different number. Also consider whether you want to change the `PoolOperatorRequired errors`.

Customer's response: We will update the 101 errors and combine the two

`PoolOperatorRequired` errors into one.

L-11 Missing check on `index`		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
File: tranche_vault - contract.rs	Status: Fixed	

The function initialize() of TrancheVault doesn't check that index is within the range [0,1] (e.g. - JUNIOR_TRANCHE or SENIOR_TRANCHE), while the Solidity code does have a check for this in initialize() of TrancheVault.sol:

```
JavaScript
if (seniorTrancheOrJuniorTranche > 1) revert Errors.InvalidTrancheIndex();
trancheIndex = seniorTrancheOrJuniorTranche;
```

Exploit Scenario: Call function initialize() with a value of 7 for `index`.

Customer's response: We will fix this.

Informational Severity Issues

I-01. Enum element `ProtocolOn` not used

Description: File `storage.rs` contains `enum StorageKey`, which contains the element `ProtocolOn`. This element `ProtocolOn` isn't used. Instead `home_config::read_is_paused` is used to check the protocol is `on`.

Consider removing the element `ProtocolOn`.

Customer's response: We previously cached the protocol state in `pool_storage` but then removed it. Looks like we forgot to remove the storage key. We will remove it.

Certora: This is fixed

I-02. Comment `make_principal_payment()` incorrect

Description: The function make_principal_payment() shows the following comment:

```
JavaScript
// # Access Control
// Only the borrower and the Sentinel service account can call this function.
```

However the authorization check only allows the `borrower` to call the function.

Customer's response: We originally had both, but then decided that it's unlikely for the Sentinel to call it, so we removed that but didn't update the comment. We'll update the comment.

Certora: This is fixed

I-03. Comment `process_redemption_requests()` outdated

Description: The following comment in process_redemption_requests() doesn't seem relevant anymore:

```
JavaScript
// Calculate senior LP token prices.
// In a uni-tranche pool, the senior tranche is disabled, so the senior supply will be 0.
// Set the senior token price to 0 if that's the case.
```

Customer's response: We will remove the comment.

Certora: This is fixed

I-04. Comment `get_days_diff()` outdated

Description: The comment in get_days_diff() seems outdated, because there is no logic to detect the `startDate` is 0.

```
JavaScript
// Returns the number of days between the two given dates. If startDate is 0, then
// use the current block timestamp as the start date.
```

Customer's response: Yes it's outdated. We'll remove it.

Certora: This is fixed

I-05. Function `withdrawable_assets()` could use `lrr.withdrawable()`

Description: The file `tranche_vault.rs` contains the function `withdrawable_assets()` with the following code:

```
JavaScript
pub(crate) fn withdrawable_assets(env: &Env, lender: &Address) -> u128 {
    ...
    let mut assets = lrr.total_amount_processed() - lrr.total_amount_withdrawn();
    ...
}
```

The function `lrr.withdrawable()` can also be used because it calculates the same value, although it uses `saturating_sub()`.

Customer's response: We can use `lrr.withdrawable()`. We don't expect this to panic anyway.

Certora: This is fixed

I-06. Function `drawdown()` could return data

Description: The function drawdown() doesn't return any value. The comparable function drawdown() in Solidity does return `netAmountToBorrower`. For consistency consider letting `drawdown()` also return `net_amount_to_borrower`.

Customer's response: We are currently not using that return value, but we can add it for consistency.

Certora: This is fixed

I-07. Function `get_amount_available_for_drawdown()` could check more

Description: The function get_amount_available_for_drawdown() doesn't check thoroughly if drawdown is (still) possible. It could also check the following:

- cr.remaining_periods == 0
- cc.revolving
- state Approved or GoodStanding
- being late with payments

Customer's response: This function is provided as "view-only" for the user to know how much they could potentially borrow, so we didn't want to make it super complex.

From a risk perspective, It's not used in any other functions within the contract. If the user attempts drawdown with the amount while the bill is not in a valid state then `drawdown()` would revert, so no harm will be done.

I-08. Steller/Rust and Solidity code differs slightly

Description: The function make_payment() is slightly different than the comparable Solidity version makePayment(). The result is the same, but the Rust/Steller function is simplified. These differences make maintenance and code review more difficult.
Consider keeping the code bases as similar as possible, for example if a new release is made of any of the code bases.

```
JavaScript
pub(crate) fn make_payment(...) ... {
    ...
    if remaining >= cr.total_past_due {
        ...
    } else {
        cr.total_past_due -= remaining;
        ...
    }
}
```

```
JavaScript
function _makePayment(...) ... {
    ...
    if (amount >= cr.totalPastDue) {
        ...
    } else {
        ...
        cr.totalPastDue -= uint96(
            paymentRecord.yieldPastDuePaid +
                paymentRecord.principalPastDuePaid +
                paymentRecord.lateFeePaid
        );
    }
}
```

Customer's response: We will change this in the Solidity code in the future.

I-09. Function `make_principal_payment()` could use `<=`

Description: The function make_principal_payment() has code with `<` while the comparable Solidity function _makePrincipalPayment() has code with `<=`.

```
JavaScript
if timestamp < cr.next_due_date {
    ...
```

```
}
```

This doesn't make much difference, but for consistency it might be better to keep them the same.

Customer's response: It doesn't make much difference indeed, but we can update to be consistent

Certora: This is fixed

I-10. Inconsistent error codes

Description: The function approve_credit() uses the error code `CreditNotInStateForApproval`.

```
JavaScript
pub(crate) fn approve_credit(
    ...
    if !matches!(cr.state, CreditState::Deleted | CreditState::Approved) {
        panic_with_error!(env, CreditManagerError::CreditNotInStateForApproval);
    }
    ...
}
```

However the equivalent Solidity function _approveCredit() uses `CreditNotInStateForUpdate` in the comparable location.

Customer's response: Either is fine, but we'll probably stick with `CreditNotInStateForApproval` since it's more specific.

I-11. Newer version of library chrono is available

Description: According to Cargo.toml, version 0.4.31 of the library chrono is used. Function to_utc_datetime() uses `NaiveDateTime::from_timestamp_opt()`, which is deprecated according to the Rust documentation.

```
JavaScript
pub fn to_utc_datetime(timestamp: u64) -> DateTime<Utc> {
    NaiveDateTime::from_timestamp_opt(timestamp as i64, 0)
        .unwrap_optimized()
        .and_utc()
}
```

Consider upgrading to a newer version of `chrono`. In that case the function `and_uct()` would not be necessary.

Customer's response: As long as the latest version doesn't break anything and there are no major known issues, we can upgrade.

Certora: This is fixed

I-12. Rust function to calculate the number of days could be used

Description: Function get_days_diff_since_previous_period_start() calculates the number of days by dividing a period by `SECONDS_IN_A_DAY`.

```
JavaScript
pub fn get_days_diff_since_previous_period_start(...) {
    ...
    ((timestamp - previous_period_start) / SECONDS_IN_A_DAY) as u32
}
```

This might round down if a subtracted leap second would be applied. Luckily leap seconds are only added so this doesn't occur. However, to be safe, the Rust function `num_days()`could also be used. Here is a snippet to show this:

```
JavaScript
//https://play.rust-lang.org
use chrono::{ Datelike, TimeZone, Utc};
fn main() {
   pub const SECONDS_IN_A_DAY: u64 = 86_400;
```

```
    let date1 = Utc::now();
    let date2 = Utc.with_ymd_and_hms(date1.year(), date1.month(), 1, 0, 0, 0).unwrap();
    let timestamp1 = date1.timestamp() as u64;
    let timestamp2 = date2.timestamp() as u64;
    let dd1 = ((timestamp1 - timestamp2) / SECONDS_IN_A_DAY) as u32;
    let dd2 = (date1 - date2).num_days();
    println!("days {} {}", dd1,dd2);
}
```

Customer's response: This solution sounds good.

Certora: This is fixed

I-13. Clearer comments

Description: The comments for DepositRecord could be made somewhat more clear:
The comment `last_deposit_time` could be extended by `"the lender"`.

```
JavaScript
pub struct DepositRecord {
    // The total amount of underlying tokens deposited by the lender
    pub(crate) principal: u128,
    // The last deposit time in this pool
    pub(crate) last_deposit_time: u64,
}
```

The comments for LenderRedemptionRecord could be made somewhat more clear:
There is a typo: "fro" => "for".

In function add_redemption_request() there is a typo:
`asserts_after_redemption` ⇒ `assets_after_redemption`

```
JavaScript
pub(crate) fn add_redemption_request(..) {
    ...
    let asserts_after_redemption = convert_to_assets(env, total_shares - shares);
    ...
}
```

Function refresh_yield_tracker() is empty and seems irrelevant. However it is supposed to be overridden by individual policies. It would be helpful to add a comment about this.

```
JavaScript
// Refreshes the amount of assets and unpaid yield for the senior tranche.
...
fn refresh_yield_tracker(&self, _env: &Env, _assets: &TrancheAssets) {}
```

Customer's response: For `DepositRecord` : It should be clear from the comment for `principal` that this struct is for individual lenders, but we can add that as you suggested to be consistent.

For `LenderRedemptionRecord`: We also noticed the typo and have changed it locally. Going to submit a PR for it.
For asserts_after_redemption: Will fix the typo

For refresh_yield_tracker: It should be pretty clear from an inheritance perspective that the empty body signifies that the default implementation does nothing, and policies implementing this trait should override this function as they see fit. But we decided to add the comment as suggested to make things clearer.

Certora: This is fixed

I-14. Confusing name for State

Description: The `CreditState::Deleted` seems like an end-state. However after this, a loan can be started again for the same borrow via `approve_credit()`.
This is not obvious for someone reading and understanding the code.

We would recommend changing this to something like `CreditState::Idle`.

Customer's response: This makes sense, but we probably won't make changes during this iteration since we may have to change a bunch of different places with a low ROI.

Appendix A

On 13/11/2024, Certora was asked by Huma to conduct an audit of a further code change:

Project Name	Repository (link)	Commit Hash	Platform
Huma protocol	https://github.com/00labs/huma-soroban-contracts	89d1b6	Stellar

The purpose of the change (see PR #160) is to allow the Sentinel Service to submit redemption requests on behalf of lenders. The Certora team found no further issues to report.

Appendix B

On 9/12/2024, Certora was asked by Huma to conduct an audit of an additional code change::

Project Name	Repository (link)	Commit Hash	Platform
Huma protocol	https://github.com/00labs/huma-soroban-contracts	aef8c76	Stellar

The purpose of the change (see PR #163) is to allow certain lenders to withdraw their yields instead of reinvesting them, as well as a smaller change related to the Soroban SDK version (see PR #164). The Certora team found no further issues to report.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora 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.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.