SMART CONTRACT AUDIT

Certification

DECENTRALIZED LEVERAGE TRADING CONTRACT

SMART CONTRACT AUDITS
Mar 16th, 2024 / v.0.2
Audited source code version:
1d54b371cd4b3f0a34d65bd38682f9350f859062

Structure and Organization of the Document

Some sections are more important than others. The most critical areas are at the top, and the less critical sections are at the bottom. The issues in these sections have been fixed or addressed and will show by the "Resolved" or "Unresolved" tags. Each case is written so you can understand how serious it is, with an explanation of whether it is a risk of exploitation or unexpected behavior.

CRITICAL

These issues can have a dangerous effect on the ability of the contract to work correctly.

HIGH

These issues significantly affect the ability of the contract to work correctly.

MEDIUM

These issues affect the ability of the contract to operate correctly but do not hinder its behavior.

LOW

These issues have a minimal impact on the contract's ability to operate.

INFORMATIONAL

These issues do not impact the contract's ability to operate.

Issues

1. Loss of funds

Fixed / CRITICAL

Description: The collateral is used in trade. This can, in multiple scenarios, lead to loss of funds for lenders, because their loan is not backed by anything. For example, if a stablecoin is borrowed to be traded for a regular coin and the coin value drops fast, the lenders will lose money (if the liquidation happens too late). It is expected, the lending is not guaranteed to be profitable, but lenders must be compensated for their loss at least with the value of the collateral initially deposited by the borrower. In the current implementation, the lender can lose all his lended tokens.

! Possible fix to research

```
Keep the collateral in the contract. For a 2:1 leverage position, a
borrower should get 2 tokens available for trading, backed by a 1 token
collateral. In the current implementation, he borrows 2 and also uses the
one he deposited in the trade, essentially leading to a 3 tokens trade
position. For a 1:1 position, there should be essentially no tokens being
borrowed.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

2. Funds draining

Fixed / CRITICAL

The platform profit is used to account for the lenders eventual losses. In case a trade is not successful, any losses that a lender might be exposed to (the initial amount deposited and the lending fees) are being assured by the platform. While the intention is noble, this model is not economically feasible. There is nothing that stops a malicious player from creating multiple lending and borrowing offers (even from the same address - there's no check that disallows one to do it) high maximum fees, match them into borrowing positions and drive them into being unsuccessful (by manipulating the liquidity pools for example). When closing, potentially huge lending fees (from pre configured maximum fee percents) might be extracted by the malicious player.

! Possible fix to research

```
Don't use the platform funds to account for the user losses. A losing
position that is not liquidated at the right time might result in losses
for the lender too. It can happen in leverage trading. Due to malicious
players placing both lending and borrowing offers (either from the same
address or from different one), compensating lenders might be dangerous.
It's better to enforce the means not to get into that situation (by,
for example, not using the borrower's collateral in the trade, having
different liquidation thresholds for different pairs, using safe prices
instead of spot ones, and so on).
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

3. Vulnerable to flash loans

Fixed / CRITICAL

Description: The contract is vulnerable to flash loans. There's no checks when a trade is made, the contract reads the current price on a DEX and executes the trade at that current price. A malicious player can intercept such calls and sandwich attack them by making a flash loan. In the same block as the trade, it can manipulate the price before and after the trade, resulting in loss of money for lenders and borrowers.

! Possible fix to research

```
Use a safe price instead of the current price or add certain slippages or
barriers such that it protects the parties involved in a trade.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

4. Unable to close position

Fixed / HIGH

Description: The contract panics if there are not enough funds to cover for the lenders (initial amount and fees), hence a borrower might not be able to close his position in order to minimise a supposed not successful trade which can lead to even more losses for him and the lenders.

! Possible fix to research

```
Introduce a mechanism that minimises losses (e.g.: an automatic
liquidation mechanism). The current implementation is dependent on the
owner to save the situation by adding more funds that the contract will
use to compensate the lenders.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

5. Blocked lender funds

Fixed / HIGH

Description: The endpoint's name, **'closeOffer'**, is quite misleading, it should essentially be **'closeUnopenedBorrowPosition'**, since is callable only the by borrower and the SC owner and it closes a Borrower Position (alongside with the adjent Lender Positions, Debt Positions). The lender in this case has no power in closing the order if the max duration is surpassed. If the contract owner will not monitor, observe and close the order, the borrower can keep the funds locked for as long as he wants.

! Possible fix to research

```
If max duration is surpassed, any lender that participates in the trade
should be able to close the position even if the trade has not yet been
done.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

6. Blocked trades

Not applicable / HIGH

Description: The contract is pausable. During the pause, offers cannot be made, trades cannot be opened. The problem is that the trades cannot be closed too, and the lender fees accumulate. Traders that want to close their position during this time cannot do it and when the pause is over and he closes his trade (assuming can & wants to close it) will be paying fees for the whole duration of the trade.

! Possible fix to research

```
Mark the trade as closed and save the block nonce such that even if a
trade cannot be closed, in case for example a maintenance is being done,
the borrower won't have to pay lender fees for the whole pause duration.
```

! Response

```
Expected, we will allow the contract owner to close trades in case of
emergencies.
```

! Status

```
Accepted & Closed.
```

7. Expired positions

Fixed / HIGH

Description: A Borrower Position can be opened even if the **'end_block_nonce'** crossmark has passed. Lenders can opt to close or liquidate the Position immediately after but at that point damage could have already been done (opening the position involves making the trade / **'swap'**).

! Possible fix to research

```
Deny opening of positions if they are already expired.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

8. Out of gas / read operations

Fixed / HIGH

Description: When querying **'getLendingOffers'** and **'getBorrowingOffers'**, the execution can easily run out of gas / read operations. This can be problematic when trying to sync the SC storage with a backend offer matcher or liquidator engine.

! Possible fix to research

```
The quick fix is to add pagination to the functions, 'offset' and 'limit'
arguments, in order to have access to all offers, not just the ones in
the front of the queue. A complete fix should include an atomic way to
get them, such an implementation might be possible by getting all the
contract storage (key-value json) and manually parsing, although the
solution might require testing and validation for huge amounts of offers.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

9. Adding collateral to expired position

Not applicable / MEDIUM

Description: There's no check to verify if a position is expired when trying to add collateral to it.

! Possible fix to research

```
Add a check to see if the position is expired.
```

! Response

```
As designed, the lender can always close the position if he wants when
the position is expired.
```

! Status

```
Accepted & Closed.
```

10. Unused function

Fixed / LOW

Description: The **'require_can_remove_borrowing_offer'** internal function is unused.

! Possible fix to research

```
Use it in 'closeOffer' or delete it.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

11. Clippy warnings

Fixed / LOW

Description: There are a few places where double reference warnings happen and a few others, for example in the **'create_borrowing_offer'** function.

! Possible fix to research

```
Find the warnings using 'cargo clippy' and fix them.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

12. Typo

Fixed / LOW

Description: The view function name above the **'platform_fee_percent'** is **'getSlippagePercent'**.

! Possible fix to research

```
Rename it to 'getPlatformFeePercent'.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

13. Correctness when rounding

Fixed / LOW

Description: The functions **'calculate_weighted_average'** and **'calculate_weighted_average_substract'** round down the calculated value. Allowing for any value in between **[0, max_value]** might be a problem when matching offers configured fee values.

! Possible fix to research

```
Allow fees only in [0, max_value] but also make a check that they are
more round values ('value % DENUM == 0). Might also be a good idea to
require the value coming out of the weighted average calculus to be this
way, although it should be tested and tried.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

14. Confusing error messages

Fixed / LOW

Description: There are certain places where the same error message is returned in case a tx fails, for example '**Can not close the position at this time**'.

! Possible fix to research

```
Even if the general reason is the same, try including more details about
the particular error spot, for example  'Can not close the position at
this time, cannot pay lender fees'.
```

! Response

```
Fixed.
```

! Status

```
Accepted & Closed.
```

Verification Conditions

1 Admin functions are marked using '#[only_owner]'

```
#[only_owner]
#[endpoint(setPlatformFeePercent)]
fn set_platform_fee_percent(&self, fee: u32) {
```

2 Only whitelisted tokens and pairs are allowed

```
require!(
    !self.traded_tokens(caller).contains(&token),
    "You already have a trade position open for the token"
);
require!(
    !self.traded_tokens(caller).contains(&token_out),
    "You already have a trade position open for the token out"
);
```

3 Removing offers can be done by either the offer owner or the admin

```
require!(
    is_contract_owner || is_offer_owner,
    "Only offer owner can remove offer"
);
```

4 One cannot hold multiple trades open for the same token

```
require!(
    self.trade_position(&caller, &token).is_empty(),
    "Can not borrow more assets for a token while a trade
    position is open"
);
```

Suggestions (Optional)

1. Use Framework versions
Description: Use latest Framework Version and Testing Framework Version when deploying the mainnet project.

```
Response: Fixed.

Status: Accepted & Closed.
```

2. Structures good practices
Description: It is a good idea to include the **'id'** in the structure identified by an **'id'**. Also, store timestamps such as creation, last update and so on. Also, do not delete the offers or positions once they expire or are removed or already used. Mark them using a flag. Those suggestions will make any debugging / reverse engineering of any situation more straightforward.

3. Implement default functions
Description: Implement **'default()'** functions for **'LenderPosition**' and 'BorrowerPosition' and use them in **'add_lender_position' and 'add_borrowing_position'** to make the code more readable and clean.

4. Additional Checks
Description: When adding a new pair address, several checks can be made in order to make sure that the introduced arguments are correct. For example, the endpoint can call that contract and validate that it indeed handles the given input tokens.

5. Offer Incentives for Liquidators
Description: There's no prize for someone that helps the lenders by liquidating someone's borrow position. Would be more healthy for the project if the good actors are rewarded somehow.

6. Max duration can be 0
Description: The max duration parameter passed by a user when creating a new offer can be 0 (the value is checked only for the max value allowed). Although it does not seem to be a problem, it does not make that much sense to allow for 0 value because the position can be closed soon after its creation (after only one block).

7. Views naming
Description: Use camel case for all public endpoints and views. There are storages that have only **'#[view]'** on top of them, without a given name, and the name will be the name of the storage (snake case).

```
Response: Fixed.

Status: Accepted & Closed.
```

8. Mismatch in fees
Description: The code requires that the lending and borrowing offers have the same fee percentages to match them. Theoretically, the case where the lending fees are lower than the borrowing fee (meaning that the borrower is willing to pay even more than the lender wants) should be considered also (although it might introduce additional complexity)

```
Response: Fixed.

Status: Accepted & Closed.
```

Test results

```
running 27 tests
test create_new_borrowing_offer_validation ... ok
test create_new_offer_validation ... ok
test cancel_borrowing_offer ... ok
test cancel_lending_offer ... ok
test close_offer_end_passed ... ok
test create_and_fill_1borrowing_1lending ... ok
test match_and_fill_1borrowing_1lending ... ok
test add_collateral ... ok
test create_and_fill_1lending_1borrowing ... ok
test match_and_fill_1borrowing_1lending_different_token ... ok
test match_and_fill_1borrowing_1lending_leverage ... ok
test close_offer ... ok
test match_and_fill_1borrowing_1lending_wrong_max_duration ... ok
test match_and_fill_1borrowing_1lending_different_fee_percentage ... ok
test match_and_fill_1borrowing_2lending_rounding ... ok
test match_and_fill_1full_borrowing_1partial_borrowing_1full_lending ... ok
test match_and_fill_1borrowing_1lending_twice_same_borrower_duration_more ... ok
test match_and_fill_1full_lending_1partial_lending_1full_borrowing ... ok
test match_and_fill_1lending_1borrowing_leverage ... ok
test match_and_fill_1borrowing_1lending_twice_same_borrower_duration_less ... ok
test match_and_fill_2borrowing_1lending ... ok
test match_and_fill_2lending_1borrowing ... ok
test match_and_fill_partial_2lending_1borrowing_rounding ... ok
test match_and_partialy_fill_2full_borrowing_1partial_lending_different_fee_percentage ... ok
test match_and_partialy_fill_2full_lending_1partial_borrowing ... ok
test match_and_partialy_fill_2full_borrowing_1partial_lending ... ok
test match_and_fill_partial_6lending_1borrowing_rounding ... ok

test result: ok. 27 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.03s
```

```
running 10 tests
test open_position_no_pair ... ok
test open_position_expired ... ok
test borrow_and_open_position_xexchange ... ok
test open_position_token_to_token3_onedex ... ok
test liquidate_position_add_collateral_no_liquidate ... ok
test open_position_token_to_token2_xexchange ... ok
test liquidate_position_add_collateral_can_liquidate ... ok
test close_position_loss_add_collateral ... ok
test liquidate_position_and_claim ... ok
test close_position_profit_add_collateral_and_claim ... ok

test result: ok. 10 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.02s
```

Audited source code version:
546d0ae60cc419ec66521f892456b0a5e6d01359

Second source code version:
1d54b371cd4b3f0a34d65bd38682f9350f859062