SECURITY AUDIT REPORT

AshPerp trading (2)
MultiversX smart contract

by ARDA

on June 1, 2024

Table of Contents

Disclaimer **3**

Terminology **3**

Objective **4**

Audit Summary **5**

Inherent Risks **6**

Code Issues & Recommendations **9**
C14: Cannot unregister trades for a frozen trader 9

Test Issues & Recommendations **11**

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's design, which are a subset of all the possible risks. **No inherent risk doesn’t mean no risk.** It only means that no risk inherent to the code's design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the upgradability of the code might also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Audit Summary

Initial scope

- **Repository:** https://github.com/ashswap/ash-tinder-sc
- **Commit:** `e2647d578a3e294f2623cf13156bc1730884b7ec`
- **MultiversX smart contract path:** ./contracts/trading/

Final scope

- **Repository:** https://github.com/ashswap/ash-tinder-sc
- **Commit:** `37feaa4cba1c0f224d39c86ec8ae08d95ad04a82`
- **MultiversX smart contract path:** ./contracts/trading/

10 inherent risks in the final scope

1 issue in the final scope

23 issues reported in the initial scope and 1 remaining in the final scope:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	1	0	0	0	0	0
Major	5	0	0	0	0	0
Medium	9	1	0	1	0	0
Minor	7	0	0	0	0	0

Inherent Risks

R1: Users might not obtain their due profit when they close their position.

This is because the Vault might fail to pay the profits e.g. in the following cases:

- The vault's funds are insufficient to pay the profits.
- We have already reached the maximum allowed daily loss of the Vault, a parameter which is chosen by the owner.

R2: Users might not be able to close their position and would have to continue to pay the borrowing fee.

This is because order execution is done by an oracle bot to close orders that, for example, (1) can be inactive or (2) unable to provide the information necessary to close orders.

R3: The borrowing fee per block might increase over time.

This is because:

- The fee per block can be changed by the admins at any time.
- The fee per block increases as the gap between the open interest of LONG positions and SHORT positions widens.

R4: Users might not get their take profit executed whereas the execution price was met and therefore would lose the expected

profit.

This is because order execution is done by an oracle bot that, for example, (1) could miss the moment when the execution price was met or (2) could choose a price worse than the market price.

R5: Users might not get their stop loss executed as soon as the execution price is met and therefore would lose more than expected.

This is because order execution is done by an oracle bot that, for example, (1) could miss the moment when the execution price was met or (2) could choose a price worse than the market price.

R6: Users might get liquidated whereas the liquidation price has not been met yet.

This is because liquidations are executed by an oracle bot that could choose a price worse than the market price.

R7: Users might not get their limit order and stop order executed whereas the execution price was met.

This is because order execution is done by an oracle bot that, for example, (1) could miss the moment when the execution price was met or (2) could choose a price worse than the market price.

R8: Users pay open fees even if their order could not be opened, and pay closing fees even if their order could not be closed.

This is because the oracle bot always takes the fee, no matter if it successfully fulfilled the user's request to open or close the order. For example, failures in opening the order can happen if the final price exceeds the user's slippage, if the total open interest exceeds the maximum allowed value on that market, of if the market is paused.

R9: The owner can cancel trades anytime without traders or liquidity providers earning profits.

This is because the owner has the ability to cancel any open trade, and in that case, the contract will send back the collateral to the trader without computing the profits or losses of the trade.

R10: Users might not obtain the expected fee discount and rebate.

This is because the calculation of trading volumes is done by external trusted contracts which might perform wrong calculations, and these volumes determine the discounts for NFTs owners and referred traders, as well as the rebates for referrers.

Code Issues & Recommendations

Since the code is not open-source, only the remaining issues are published.

C14: Cannot unregister trades for a frozen trader

Severity: Medium **Status:** Won't fix

Location

```
contracts/trading/src/callback.rs
   _unregister_trade
```

Description

Current behavior: If the trader’s wallet is frozen for a specific token and is in the same shard as the contract, then the contract will fail to send back the tokens to the trader when closing the order. This means:

- The trade cannot be unregistered, in particular liquidated, thus the Vault would not get her expected profits.
- The owner cannot call `force_close_trade_market`.

Expected behavior: If a trader is frozen for a token and is in the same shard as the contract, then the contract should not try to send the tokens to him. As this is the case where the transaction would fail, and so trade cannot be liquidated and the owner cannot execute `force_close_trade_market`.

Note that the transaction failure only happens if the trader is both frozen and in the same shard, for example if his wallet is in another shard, the transaction would not fail.

Recommendation

We recommend checking if the trader is frozen for the token that is going to be sent and if he is in the same shard as the contract. In that case, we recommend not sending back funds to the trader with that token because it would block the contract and it is the trader’s responsibility to be able to receive transfers.

Verifying whether an account is frozen and in the same shard can be done as in the MultiversX Bridge ETH smart contract:

```
fn is_account_same_shard_frozen(&self, sc_shard: u32,
  dest_address: &ManagedAddress, token_id: &TokenIdentifier
  ) -> bool {
  let dest_shard =
self.blockchain().get_shard_of_address(dest_address);
  if sc_shard != dest_shard {
    return false;
  }

  let token_data = self
    .blockchain()
    .get_esdt_token_data(dest_address, token_id, 0);
  token_data.frozen
}
```

Test Issues & Recommendations

Since the code is not open-source, only the remaining issues are published.