TORNADO DIVERSIFICATION PROPOSAL AUDIT

Distributed Lab

By Artem Chystiakov and Armen Arakelian, JUNE 2022

Summary: production-ready.

The audit was made on the entire treasury-diversification-proposal repository including the 1inch limit order integration. We have done due diligence on the previous governance upgrades to make sure the new upgrade is compatible, as well as the 1inch limit order contract to check the correctness of the integration.

The overall code structure is excellent and readability is decent. Every bug that was found during the former audit session is either fixed or made clear of the correctness of the business logic behind it. We think that the contracts are production-ready.

The audit commit hash:

d064c4f2763f0ff3b9c586a55b1c4d8697d5d738

The reaudit commit hash:

ff8e449e0d8750fbfc46065cb1f208d59399f610

STRUCTURE AND ORGANIZATION OF DOCUMENT

The information about the severity of the bugs is given below. The list starts with the description of the critical bugs in red and ends with the informational ones in blue.

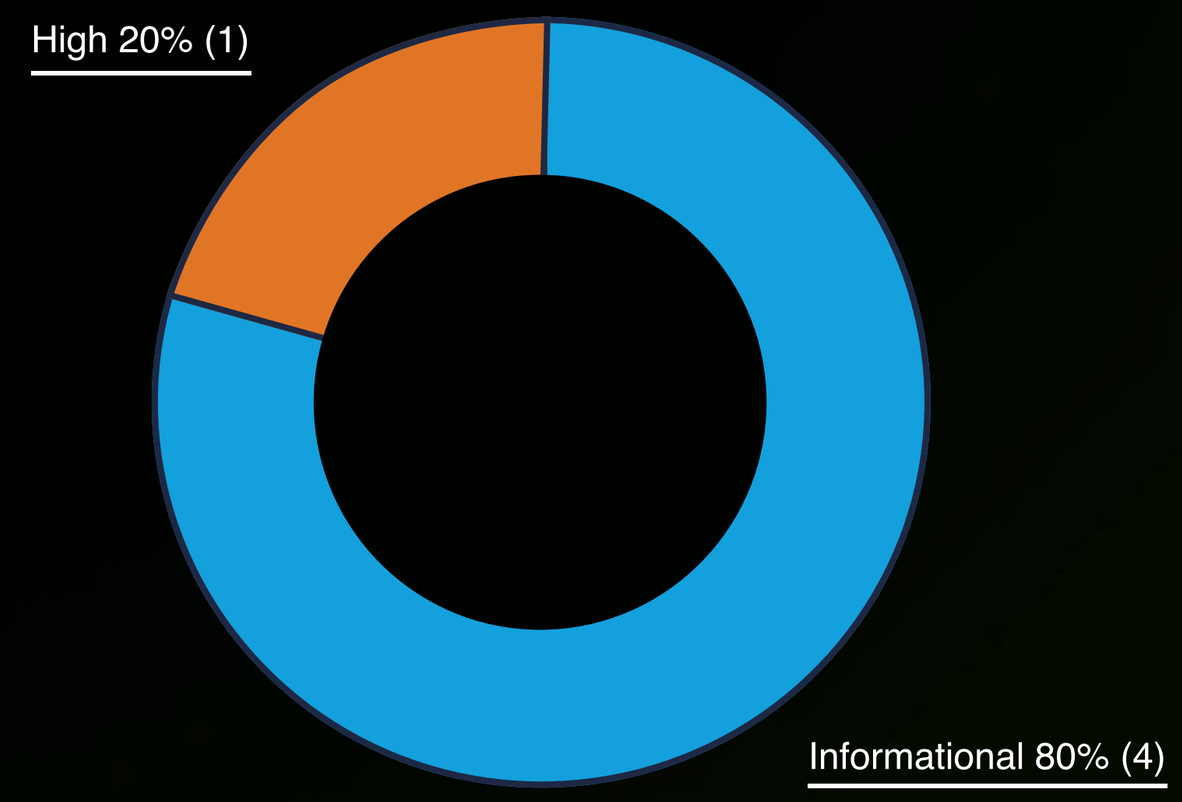
crit **Critical** - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

high **High** - The issue affects the ability of the contract to compile or operate in a significant way.

med **Medium** - The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

low **Low** - The issue has minimal impact on the contract’s ability to operate.

info **Informational** - The issue has no impact on the contract’s ability to operate.



01 UNISWAPV3ORACLEHELPER.SOL

Unused library

The contract imports `LowGasSafeMath` library but never uses one.

```
library UniswapV3OracleHelper {
     using LowGasSafeMath for uint256;

     . . .
}
```

Recommendation:

Remove the library from the contract.

Status: fixed

02 GOVERNANCEDIVERSIFICATIONUPGRADE.SOL

The contract forbids locking tokens for users that already have locks

The function `lockWithVestingTo()` that is called from the `SaleHandler` contract does not work with the “old” platform users that already have tokens locked in the governance. We believe that this limitation significantly decreases user experience and should be disabled.

```
function lockWithVestingTo(
     address beneficiary,
     uint256 amount,
     uint256 timestamp
) public virtual onlyVestingHandler {
     require(lockedBalance[beneficiary] == 0, "Beneficiary already has
          locked tokens");

     . . .
}
```

Recommendation:

1. Remove the limitation
2. Add `updateRewards(beneficiary)` modifier to the function
3. Add the `amount` to `lockedBalance[beneficiary]` instead of assigning it
4. Take the maximum of `timestamp` and `canWithdrawAfter[beneficiary]` before passing the value to the `_lockTokens()` function

Status: expected behavior

As discussed with the client, current realization is the expected behavior of the contract. The intent behind this decision is to secure users from accidentally locking their existing funds in the governance contract for way too long.

03 SALEHANDLER.SOL

Inconsistent usage of variables (code readability)

The `function _transfer()` uses `sender` and `msg.sender` throughout the function. These two variables mean mostly the same thing and only one of them should be used to improve code readability.

```
function _transfer(
     address sender,
     address recipient,
     uint256 amount
) internal virtual override {
     if (recipient == GOVERNANCE) {
          require(sender == msg.sender, "SH: TransferFrom to
               GOVERNANCE is restricted");
          _burn(sender, amount);
          if (block.timestamp >= endVestingDate) {
               require(ERC20(TORN).transfer(msg.sender, amount), "SH:
                    failed to send TORN");
          } else {
               IDiversificationUpgrade(GOVERNANCE).lockWithVestingTo(
                    msg.sender, amount, endVestingDate);
          }
          return;
     }

     . . .
}
```

Recommendation:

Use `sender` throughout the function to improve code readability.

Status: fixed

03 SALEHANDLER.SOL

Useless burn call

In the `finalizeSale()` the `_burn()` function might be called with the 0 value.

```
function finalizeSale() external {
     . . .

     uint256 unsoldTokens = balanceOf(address(this));
     if (unsoldTokens > 0) {
          require(ERC20(TORN).transfer(GOVERNANCE, unsoldTokens), "SH:
               failed to send TORN");
     }

     _burn(address(this), unsoldTokens);
}
```

Recommendation:

Move the `_burn()` function call under the if above.

Status: fixed

03 SALEHANDLER.SOL

The price discount does not include the market depth

The price of `TORN` is taken from the UniswapV3 protocol via the oracle library and there are no problems with that. However, the price is only considered for the `1` standalone token and then multiplied by the sale amount. This approach has a core flaw because it does not consider the market depth. With this being said, let's imagine the extreme case where the whole `120000 TORN` are sold to a single buyer. If the Uniswap spot price is `0.015 ETH` for `1 TORN`, the total value is `120000 * 0.015 = 1800 ETH respectively`. In contrast, if the same whale went to the Uniswap and market-bought the same amount of `TORN`, he would spend roughly `6500 ETH` (due to lack of liquidity). The "discount" is `72%`, which is way above the desired `20%` set by the governance. We think that such a discount would attract quite some whales who would buy all of the available tokens and then dump them on the exchanges (after the vesting period expiration), weakening the economy of the protocol.

Recommendation:

There is no direct solution to this vulnerability, however here are some recommendations that might smooth the impact.

1. Reduce the discount to account for the market depth.
2. Implement the vesting scheme with the cliff period for the big amounts.

Status: expected behavior

As negotiated with the client, this selling strategy is indeed the way most of the "off-market" trades are made and exclusion of the market depth is absolutely fine. The client is aware of the "extra discount" burden behind this business logic.

With great appreciation and respect.

Distributed Lab

JUNE 2022