Turbo Security Audit

Report Version 1.0

November 22, 2024

Conducted by **Hunter Security**

Table of Contents

1 About Hunter Security **3**

2 Disclaimer **3**

3 Risk classification **3**
3.1 Impact . . . 3
3.2 Likelihood . . . 3
3.3 Actions required by severity level . . . 3

4 Executive summary **4**

5 Findings **5**
5.1 High . . . 5
5.1.1 Incorrect amount of turbo sent to treasury and burnt . . . 5
5.2 Medium . . . 5
5.2.1 Tokens may get permanently stuck in the Auction when minting liquidity . . . 5
5.3 Low . . . 6
5.3.1 Potentially unfair distribution of token in an edge case . . . 6
5.4 Informational . . . 6
5.4.1 Typographical, non-critical or centralisation issues . . . 6

1 About Hunter Security

Hunter Security is an industry-leading smart contract security auditing firm. Having conducted over 100 security audits protecting over $1B of TVL, our team always strives to deliver top-notch security services to the best DeFi protocols. For security audit inquiries, you can reach out on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Turbo
Repository	https://github.com/De-centraX/pillar-turbo-contracts
Commit hash	f114458c437972d4e5c4c74dd35f887bce934c09
Resolution	de28e464395dcd4a1cc6b18857da6583da4d11bb
Methods	Manual review & testing

Scope

src/actions/SwapActions.sol
src/Auction.sol
src/Turbo.sol
src/TurboBnB.sol
src/TurboTreasury.sol
src/VoltBurn.sol

Issues Found

High risk	1
Medium risk	1
Low risk	1

5 Findings

5.1 High

5.1.1 Incorrect amount of turbo sent to treasury and burnt

Severity: High

Context: TurboBnB.sol

Description: The TurboBnB contract periodically buys Turbo tokens by swapping the Hyper funds sent to it by the auction. The bought tokens are then either distributed among the liquidity bonding address, the treasury contract, or burned.

The problem is that instead of sending 50% of the bought amount to the treasury, which would be then distributed in the Auction, the amount of 0.5e18 tokens (units) is sent which is most likely meant to be used as a multiplier.

```
turbo.transfer(LIQUIDITY_BONDING_ADDR, wmul(turboAmount, uint256(0.08e18)));
turbo.transfer(address(turbo.treasury()), uint256(0.5e18));
burnTurbo();
```

Recommendation: Consider implementing the following change as well as improving the test scenarios and assertions:

```
turbo.transfer(address(turbo.treasury()), wmul(turboAmount, uint256(0.5e18)));
```

Resolution: Resolved.

5.2 Medium

5.2.1 Tokens may get permanently stuck in the Auction when minting liquidity

Severity: Medium

Context: Auction.sol

Description: Auction.__addLiquidityToPool_ mints a liquidity position in the Turbo:Hyper pool using the collected Hyper tokens from the auction's deposits. After the position is created, any leftover tokens are returned to the owner of the contract (i.e. slippage difference).

The problem occurs at the second check:

```
if (amount1Added < amount0Sorted) {
    IERC20(sortedToken1).transfer(owner(), amount1Sorted - amount1Added);
}
```

The amounts compared are not of the same token (*token1*) which leads to either having tokens permanently stuck in the contract or attempting to send back to the owner a significantly higher amount than intended.

Recommendation: Consider implementing the following change as well as improving the test scenarios:

```
if (amount1Added < amount1Sorted) {
```

Resolution: Resolved.

5.3 Low

5.3.1 Potentially unfair distribution of token in an edge case

Severity: Low

Context: Auction.sol, TurboBnB.sol

Description: The *_updateAuction* function emits the planned amount of Turbo tokens for the current day when the first deposit for the day is made. In case the *daySinceStart* is > 8 and the turbo treasury contract has no balance, the function would revert, not allowing to make deposits for that day until some amount of Turbo tokens is deposited into the treasury.

The problem is that instead of blocking deposits for the whole day due to not having any Turbo tokens to distribute among depositors, the function would allow deposits again as soon as the treasury get funded.

This means that if the treasury receives its tokens towards the end of the day, there might be just a several minutes window for users to make a deposit, making the distribution unfair as significantly fewer depositors would take the same portion of the treasury's balance as for a full day time window.

Recommendation: Consider either:

- Using the time of the day left as a multiplier for the amount of Turbo tokens in the treasury to be emitted.
- Blocking deposits for the whole day after some point if there is no treasury balance and Turbo to be emitted.

Resolution: Acknowledged.

5.4 Informational

5.4.1 Typographical, non-critical or centralisation issues

Severity: Informational

Context: .

Description: The contracts contain one or more typographical mistakes, non-critical issues or centralisation vulnerabilities. In an effort to keep the report size reasonable, we enumerate these below:

1. *turboEmitted* can still be set to 0 if the treasury balance is in the range of 1 to 4 units of tokens which would allow for deposits that would gain nothing in return.
2. Consider enforcing an upper and lower limit for *intervalBetweenBurns*.
3. Forgotten repeated *hasLP* check in *auction.addInitialLiquidity*.

Recommendation: Consider fixing the above typographical mistakes, non-critical issues or centralisation vulnerabilities.

Resolution: Partially resolved.