AwesomeX Audit Report

Version 1.0

ptsanev

September 15, 2024

AwesomeX Audit Report

Plamen Tsanev

September 15, 2024

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Severity classification
- Audit Scope
- Findings
  - High
  - Medium
  - Low

Protocol Summary

AwesomeX is an hyper-deflationary perpetual compounding ERC20 token with buy-and-burn mechanics and custom tokenomics, designed for value management through controlled supply. Built on DragonX - Using TitanX to mint.

Disclaimer

This report is based on an analysis carried out within a specific scope and timeframe, utilizing the provided materials and documentation. It does not claim to identify all possible vulnerabilities and should not be viewed as exhaustive. The findings and this report are offered “as-is” and “as-available,” with no express or implied warranties. Moreover, this report does not endorse any particular project or team, nor does it assure the project’s complete security.

About ptsanev

Plamen Tsanev, or ptsanev, is a web3 security researcher with numerous top placements in competitive audits. Ranked #5 on Hats Finance’s leaderboard, he has led several audits. Plamen has responsibly disclosed vulnerabilities in live protocols, demonstrating a thorough understanding of complex codebases and their vulnerabilities. Reach out on twitter @ptsanev

Severity classification

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

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Audit Scope

The following smart contracts were in scope of the audit:

- `AwesomeXBuyAndBurn.sol`
- `AwesomeXMinting.sol`
- `AwesomeX.sol`
- `BuyAndBurnConst.sol`

Findings

High

[H-1] Incorrect update of distributed TitanX leads to an incorrect daily burn allocation

Context: AwesomeXBuyAndBurn.sol#L156-L182

Description: Every minting period a certain amount of TitanX is distributed to the BnB(Buy and Burn) contract to be allocated throughout the week, based on the given weekly allocations. However, due to the incorrect update of the `totalTitanXDistributed` at the end of every swap and burn, the total distributed TitanX for allocation would get reduced every interval. For example, we have an X amount distributed and we have a 3% allocation for the current day. This means that each of the 192 intervals would swap and burn ~0.02% of the distribution for the day. However, due to the function updating the `totalTitanXDistributed` at the end, the next time we calculate the interval allocation, instead of taking 0.02% of X, it will be 0.02% of (X - 0.02%), leading to a smaller than intended allocation. This small difference would compound for every interval and for every day of the week, leading to incorrect allocation for the whole 1 week period.

Recommendation: Do not update the `totalTitanXDistributed` snapshot at the end of every swap and burn. The `totalTitanXDistributed` should only be updated when a minting period ends and remain the same for the entire 1 week period.

Resolution: Fixed in 0503befcbf098562dec6a57141e3dc3c1df6ad15

[H-2] Incoming TitanX is never marked as distributed

Context: AwesomeXBuyAndBurn.sol#L257-L266

Description: When new TitanX comes in for distribution during the minting period, the `distributeTitanXForBurning()` function is called, which is meant to set the `totalTitanXDistributed` in order to kickstart the burning allocations for the daily intervals. However, the function never updates the `totalTitanXDistributed`, it only updates the current interval. This means that even though the contract would be holding the TitanX funds, the internal variable tracking it will remain 0, leading to no allocation for the first interval.

Recommendation: Make sure that `distributeTitanXForBurning()` updates the `totalTitanXDistributed` to the new balance via a snapshot at the end of the call. Another important layer of security would be to make snapshots weekly instead of daily and add some form of access control to `distributeTitanXForBurning()`. Since the function lacks access control intended for future integrations, it will be possible to update the `totalTitanXDistributed` to a lower value due to the swaps every interval. This will negatively impact the allocations for the week and make them incorrect.

Resolution: Fixed in 0503befcbf098562dec6a57141e3dc3c1df6ad15

Low

[L-1] `alreadyAllocated` variable is not updated

Context: AwesomeXBuyAndBurn.sol#L495-L522

Description: The `alreadyAllocated` variable is used while tracking missed intervals between different days and transferring their meant allocations to the current interval. The variable should be updated at the end of every iteration, however it is not.

Recommendation: Update the `alreadyAllocated` variable at the end of the loop

Resolution: Fixed in 0503befcbf098562dec6a57141e3dc3c1df6ad15

[L-2] Missing events for changes in important slippage parameters

Context: AwesomeXBuyAndBurn.sol#L235-L251

Description: The slippage variables for the TitanX/DragonX swap, DragonX/AwesomeX swap and for the LP are important, owner-controlled, parameters that are subject to change over the course of the compounding. Changes to such parameters should be public to the end user via proper events when they get updated.

Recommendation: Introduce an event/events for when slippage gets changed

Resolution: Acknowledged

[L-3] Edge-case distribution if there is no added liquidity can fail

Context: AwesomeXMinting.sol#L155-L187

Description: The minting contract handles distribution of incoming TitanX that mints AwesomeX. The `_distribute()` function handles the edge-case scenario where: "If there is no added liquidity, but the balance exceeds the initial for liquidity, we should distribute the difference" and checks for the difference between current TitanX balance and the inteded `INITIAL_TITAN_X_FOR_LIQ`. This safeguard mechanism is fine, however if this difference comes out to be very small, the transaction can fail at the fee calculations. This is the case, since the fees for the team, vaults, launchpad and treasury are all calculated as small percentages, rounded up . If the amount meant to be distributed is very small, the fees could end up being rounded up to be larger than the amount for distribution, leading to an underflow

Recommendation: This is too much of an edge-case scenario to introduce proper handling for, but it is something that can occur, especially if fee percentages get changed in the future, so be mindful of increasing fees.

Resolution: Acknowledged