AwesomeX PPV Security Review

Conducted by: **Georgi Trachev**

Report Version 1.0
September 17, 2024

Contents

1. About Georgi Trachev

Georgi Trachev is a smart contract security researcher. Throughout his experience he has uncovered numerous critical vulnerabilities in a wide variety of protocols. Reach out on X here.

2. Disclaimer

Smart contract security reviews are a time, resource and expertise bound effort where security researchers do their utmost to find as many vulnerabilities as they can. They are not a 100% guarantee of the security of smart contracts.

3. About AwesomeX

AwesomeX is a hyper-deflationary perpetual compounding system, built on top of DragonX – using TitanX to mint tokens. It has mechanisms for burning and minting tokens, where the cost of minting AwesomeX tokens grows over a 125-week period.

4. Risk Classification

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

4.1 Impact

High: Causes a significant loss of funds, the protocol is directly compromised

Medium: A minor amount of funds at loss, or a functionality of the protocol can be affected

Low: Unexpected behaviour in the protocol, but does not present a significant harm

4.2 Likelihood

High: Direct attack vector; cost of attack negligible, compared to funds at risk

Medium: A sophisticated attack vector that is incentivized only in some conditions but is still likely to occur

Low: Requires many unlikely assumptions or the attacker is not incentivized

4.3 Actions required by security level

High: Issue **MUST** be fixed

Medium: Issue **SHOULD** be fixed

Low: Issue **MAY** be fixed

5. Security Assessment Summary

Overview

Repository: https://github.com/Kuker-Labs/awesomex-contracts
Commit hash: d007efec711f46da7e3df13060f4ff055eb9928e

Scope

The smart contracts included in the scope of the security review:

- src/interfaces/IDragonX.sol
- src/AwesomeXBuyAndBurn.sol
- src/AwesomeX.sol
- src/AwesomeXMinting.sol
- src/const/BuyAndBurnConst.sol
- src/utils/Math.sol
- Src/library/OracleLibrary.sol

Timeline

Audit start: September 12, 2024
Preliminary report: September 17, 2024

6. Executive Summary

Throughout the security review, conducted in the period September 12-17, 2024, 5 issues were discovered.

Findings count

High Risk	1
Medium Risk	1
Low Risk	2
Informational	1

7. Findings

7.1 High Findings

[H1] *forBnB* is calculated incorrectly, causing more TitanX to be burned or the minting of AwesomeX to revert

Severity: *High*

Context: AwesomeXMinting._distribute.sol#L172

Description: In the *mint* function of AwesomeXMinting.sol *forBnB* is calculated incorrectly. As a result, *forBnB* will be given a higher value than intended by the protocol, causing the call to revert or for more TitanX tokens to be burned. This happens as currently *forBnb* is calculated as:

```
  forBnB -= titanXForLaunchpad - titanXForDragonX -
titanXForTreasury - (titanXForTeam * 2));
```

This is wrong as *forBnB* needs to be decreased by the sum of *titanXForLaunchpad, titanXForDragonX, titanXForTreasury* and *titanXForTeam.* Multiple overflows will occur in the current calculation, eventually causing *forBnB* to be more than the initial *_amount.* For example, if *_amount* is 10,000, *forBnB* will be set to 11950, instead of the intended 7350. As a result, more TitanX will be distributed to be burned, or if there are not enough TitanX tokens in AwesomeXMinting.sol, the call to *mint* will revert.

Recommendation: Consider implementing the following change:

```
forBnB -= titanXForLaunchpad + titanXForDragonX +
titanXForTreasury + (titanXForTeam * 2));
```

Resolution: Resolved.

7.2 Medium Findings

[M1] If the current cycle ID goes above 255, claiming minted AwesomeX tokens will be impossible

Severity: *Medium*

Context: AwesomeXMinting.claim.sol#L137, AwesomeXMinting._getCycleEndTime.sol#L205

Description: In the *claim* and *_getCycleEndTime* functions of AwesomeXMinting.sol the cycle ID, for which AwesomeX tokens will be claimed, is set to a uint8 parameter. This is an issue as the maximum value of a uint8 variable is 255. Thus, if the cycle ID goes above 255 both of the functions will revert, preventing users from claiming their minted AwesomeX tokens.

Recommendation: Consider implementing uint32 instead of uint8.
Resolution: Resolved.

7.3 Low Findings

[L1] When *lastBurnedIntervalStartTimestamp == 0* allocations may not be handled correctly

Severity: Low

Context: AwesomeXBuyAndBurn._calculateIntervals.sol#L482

Description: In *_calculateIntervals* if *lastBurnedIntervalStartTimestamp == 0* it is possible for allocations to be calculated incorrectly. This happens as if *lastBurnedIntervalStartTimestamp* is still 0 *dayOfLastInterval* is set to the current day. This makes the incorrect assumption that no days have passed since the start of the minting cycles.

For example, if *startTimestamp* is at weekday 4 and *_calculateIntervals* is invoked for the first time at weekday 5 (while *lastBurnedIntervalStartTimestamp* is still 0), *dayOfLastInterval* will be set to a timestamp in day 5, instead of in day 4. As a result, the allocations for the missed intervals will be accumulated as if they occurred on day 5. This is problematic as daily TitanX allocations can be significantly greater for some days of the week compared to others. The correct implementation should set *dayOfLastInterval* to the start timestamp of the minting cycles and not the current timestamp as they may be different.

Recommendation: Consider implementing the following change:

```
uint32 dayOfLastInterval = lastBurnedIntervalStartTimestamp ==
0 ? startTimestamp :
dayCountByT(lastBurnedIntervalStartTimestamp);
```

Resolution: Acknowledged.

[L2] When *lastBurnedIntervalStartTimestamp == 0* missed interval allocations may be more than intended

Severity: Low

Context:
AwesomeXBuyAndBurn._calculateMissedIntervals.sol#L538

Description: In *_calculateMissedIntervals* if *lastBurnedIntervalStartTimestamp == 0* and the time elapsed since the

start of the minting cycles is more than the *INTERVAL_TIME* the missed intervals for which allocations will be accumulated will be one more than intended. This occurs as in *_calculateMissedIntervals* *_missedIntervals* is decreased by 1 only if *lastBurnedIntervalStartTimestamp != 0.* This is insufficient as the missed intervals should also be decreased when *lastBurnedIntervalStartTimestamp == 0* and the time elapsed since the start of the minting cycles is more than the *INTERVAL_TIME.* Otherwise, in *_calculateIntervals* an additional interval will be accounted for that is not balanced out by the deduction of *_missedIntervals*.

Recommendation: Consider implementing the following change:

```
if (lastBurnedIntervalStartTimestamp != 0 || timeElapsedSince
>= INTERVAL_TIME) _missedIntervals--;
```

Resolution: Acknowledged.

7.4 Informational Findings

[I1] Account abstraction wallets cannot call *swapTitanXForAwesomeXndBurn* as the *msg.sender* must be the same as the origin of the transaction (*tx.origin*). This incompatibility with AAW may prevent honest users from interacting with the protocol.