Phoenix Security Audit

Report Version 1.0

December 10, 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 Medium . . . 5
5.1.1 Staking manager not implementing setter for the buy action state . . . 5
5.1.2 Leftover tokens when providing liquidity due to slippage . . . 5
5.2 Low . . . 5
5.2.1 Overflow may cause unexpected behavior . . . 5
5.2.2 Insufficient input validation in privileged setter methods . . . 6
5.3 Informational . . . 6
5.3.1 Typographical mistakes, non-critical issues and code-style suggestions . . . 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	Phoenix
Repository	https://github.com/De-centraX/phoenix-contracts
Commit hash	cd0b9d7612e369dc1d2be9246284c6215cf81b46
Resolution	86201ca8f4979573a28f1f13da74db3e717c8e21
Methods	Manual review & testing

Scope

src/actions/SwapActions.sol
src/staking/BaseStakingVault.sol
src/staking/BlazeStakingVault.sol
src/staking/FluxStakingVault.sol
src/staking/TitanXStakingVault.sol
src/Auction.sol
src/AuctionTreasury.sol
src/BuyAndBurn.sol
src/Minting.sol
src/Phoenix.sol
src/TitanXStakingManager.sol

Issues Found

High risk	0
Medium risk	2
Low risk	2

5 Findings

5.1 Medium

5.1.1 Staking manager not implementing setter for the buy action state

Severity: Medium

Context: TitanXStakingManager.sol

Description: The *TitanXStakingManager* is the owner of the deployed *TitanXStakingVault* contracts. It acts as a proxy for the setter functions of each vault as there is no other address that can access them.

The problem is one important setter method is missing in the manager which is responsible for setting important properties in the vaults - *changeBuyActionState*.

Recommendation: Consider implementing the missing method.

Resolution: Resolved.

5.1.2 Leftover tokens when providing liquidity due to slippage

Severity: Medium

Context: Minting.sol

Description: In *addLiquidityToInfernoPhoenixPool*, if any *phoenix* or *inferno* amount was not taken by the pool to add as liquidity, the leftover would be locked forever in the Minting contract.

Recommendation: Consider sending the remaining token amounts back to the owner if there's any leftover due to slippage.

Resolution: Resolved.

5.2 Low

5.2.1 Overflow may cause unexpected behavior

Severity: Low

Context: Minting.sol

Description: When determining what cycle we are currently at, the following calculation is made:

```
currentCycle = uint8(timeElapsedSince / GAP_BETWEEN_CYCLE) + 1;
```

The problem is that once *timeElapsedSince / GAP_BETWEEN_CYCLE* becomes > 255, the downcast will silently overflow and restart the cycles. Therefore, the returned *currentCycle*, *startsAt* and *endsAt* will be incorrect at the given timestamp.

Recommendation: Consider removing the downcast.

Resolution: Resolved.

5.2.2 Insufficient input validation in privileged setter methods

Severity: Low

Context: src/*

Description: The following methods need to have a lower or upper cap values enforced to ensure no malicious setting is possible:

- *changeIntervalBetweenBurns*, *changeStakingCooldown* - add upper limit.
- *changeSwapCap* - can be set to 0 and renounce ownership.
- *changeLpSlippage* - should have a maximum of 1e18.

Recommendation: Consider implementing stricter input validation.

Resolution: Acknowledged.

5.3 Informational

5.3.1 Typographical mistakes, non-critical issues and code-style suggestions

Severity: Informational

Context: src/*

Description: The contract contains one or more typographical mistakes, non-critical issues and code-style suggestions. In an effort to keep the report size reasonable, we enumerate these below:

1. The *require(_incentive <= 1e18);* check in *TitanXStakingManager.changeIncentive* is redundant.
2. The *startTimestamp* should never be set in the past.
3. *totalPhoenixMinted* is updated before the tokens are actually minted.
4. Consider explicitly preventing invalid IDs in *Minting.claim*.
5. *stakingCooldown* and *firstStakeMin* are never changed. Consider making *immutable* or implementing setters.
6. *GAP_BETWEEN_CYCLE* must always be >= than *MINT_CYCLE_DURATION*.

Recommendation: Consider fixing the above typographical mistakes, non-critical issues and code-style suggestions.

Resolution: Partially resolved.