marchev

Stax
Security Review

Version 1.0

Reviewed by:

Martin Marchev

Pyro

December 20, 2024

Table of Contents

1 Introduction

1.1 About Us

Martin Marchev is an independent security researcher specializing in Web3 security. With a proven track record in competitive audits and responsible vulnerability disclosures, he has contributed to the security of high-profile projects with over $1B in Total Value Locked (TVL). For bookings and security review inquiries, reach out via Telegram, Twitter or Discord.

Pyro is distinguished independent smart contract security researcher with robust track record. Over the past year, he has improved the security of many protocols, working both alone and with others. Previously with Guardian, Pyro has audited high-profile clients like Synthetix and GMX, earning him a reputation as a trusted blockchain security researcher. Learn more at https://github.com/0x3b33.

1.2 Disclaimer

While striving to deliver the highest quality web3 security services, it is important to understand that the complete security of any protocol cannot be guaranteed. The nature of web3 technologies is such that new vulnerabilities and threats may emerge over time. Consequently, no warranties, expressed or implied, are provided regarding the absolute security of the protocols reviewed. Clients are encouraged to maintain ongoing security practices and monitoring to address potential risks.

1.3 Risk Classification

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

1.3.1 Severity Criteria

- **Critical** - Severe loss of funds, complete compromise of project availability, or significant violation of protocol invariants.
- **High** - High impact on funds, major disruption to project functionality or substantial violation of protocol invariants.
- **Medium** - Moderate risk affecting a noticeable portion of funds, project availability or partial violation of protocol invariants.
- **Low** - Low impact on a small portion of funds (e.g. dust amount), minor disruptions to service or minor violation of protocol invariants.
- **Informational** - Negligible risk with no direct impact on funds, availability or protocol invariants.

2 Executive Summary

2.1 About Stax

Stax is an NFT collection and token aimed to be a one stop shop for TitanX stakers. It features auto-compounding, liquid staking and buy-and-burn mechanics.

2.2 Synopsis

Project Name	Stax
Project Type	ERC-721 & ERC-20
Repository	Stax
Commit Hash	26af8fc5...36750229
Review Period	9 Dec 2024 - 20 Dec 2024

2.3 Issues Found

Severity	Count
Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	4
Informational	2
Total Issues	**9**

3 Findings

Medium risk

[M-1] Keepers can game the buy-and-burn incentive mechanism to always receive higher fees

Context: StaxBuyBurn.sol#L70

Description: The contract implements two different incentive fee tiers: 0.3% for standard `buyAndBurn()` operations and 0.45% when there are additional X28 tokens to swap.

However, whitelisted keepers can exploit this mechanism by sending a small amount of X28 (e.g. a couple of wei) to the contract before calling `buyAndBurn()`. This artificially triggers the condition for the higher fee tier, allowing them to consistently receive 0.45% instead of 0.3% - even when there are no legitimate X28 tokens received for burning in the contract.

While this exploitation is limited to whitelisted keepers, it allows for consistent value extraction from the protocol beyond intended rewards and could result in significant excess fee payments over time.

Example Scenario:

1. Alice is a whitelisted keeper for the buy-and-burn contract
2. The contract has zero X28 tokens, meaning Alice should receive the standard 0.30% incentivization fee
3. Alice sends 10 wei of X28 tokens to the contract directly
4. Alice calls `buyAndBurn()`
5. Since the contract now has a positive X28 balance, it performs a X28/E280 swap and thus applies the increased 0.45% fee
6. Alice receives 0.45% instead of the intended 0.30% fee, successfully gaming the system

Recommendation: Consider one of the following approaches:

1. Implement a minimum threshold of X28 tokens required to qualify for the higher fee tier
2. Track the source of X28 tokens by implementing a permissioned `distributeForBurning()` function in `StaxBuyBurn` that can only be called by authorized contracts (e.g., `Stax`, `StaxVault`, `HyperStax`). The contract should maintain a separate accounting of these X28 tokens and only apply the higher 0.45% fee when these tracked tokens are present, ignoring any X28 tokens sent directly to the contract

Resolution: Fixed in 5555fcd

[M-2] Cycles may not be periodically updated

Context: StaxVault.sol#L100

Description: The `StaxVault::updateCycle` function requires manual triggering to update and advance to the next cycle.

This can lead to issues depending on ecosystem activity. Users may delay triggering cycles to gain

an advantage. They may also lead to some losses when users burn their NFTs thinking the enxt cycle has begun, before it actually is.

Example Scenario:

1. 9 days and 4 hours have passed since the start of cycle 2
2. Cycle 3 is ready to begin
3. Alice mints an NFT during cycle 2
4. Alice then calls `updateCycle`

As a result, Alice's NFT is minted in cycle 2, even though it should belong to cycle 3. Alice would incorrectly receive a portion of the rewards for cycle 2.

Recommendation: Consider automatically updating cycles during all user interactions with the NFT and Vault contracts.

Resolution: Acknowledged

[M-3] `StaxVault.updateCycle()` function is susceptible to griefing attacks

Context: StaxVault.sol#L100, StaxVault.sol#L391

Description: The `updateCycle()` function in the `StaxVault` contract is permissionless and updates the reward cycle. It swaps any available TitanX balance for X28 using the X28/TitanX Uniswap v3 pool if:

- The TitanX pool balance (`titanXPool`) exceeds the minimum threshold (`minCyclePool`)
- Native minting of X28 is disabled (X28 price deviates by more than 10% from the TitanX price)

A `_twapCheck()` ensures the function is protected against flashloan-based sandwich attacks. However, the `minAmountOut` parameter and TWAP checks introduce a griefing attack vector. Specifically, a legitimate user calling `updateCycle()` with a proper `minAmountOut` value to ensure minimal slippage, may have their transaction reverted if a malicious actor frontruns them to inflate the X28 price. This manipulation increases slippage, causing the transaction to violate the `minAmountOut` or TWAP checks, resulting in a revert. This griefing can disrupt cycle updates, affecting the protocol's operations.

Recommendation: Separate the X28 minting/swapping mechanism from the `updateCycle()` function and make the TitanX transformation process permissioned (similar to the BuyAndBurn contract). This would eliminate griefing risks and mitigate risk of MEV attacks during cycle updates.

Resolution: Acknowledged

Low risk

[L-1] User does not receive incentives in ETH

Context: HeliosStax.sol#L146-L154

Description: Currently, Helios provides rewards in ETH and TitanX. However, users claiming these rewards receive only one of the two as incentives. While the impact is small (e.g., users may delay claims), it is still important to reward users with their appropriate share of the claimed amount.

Recommendation: Consider sending incentives for all received tokens. Example:

```
        if (claimAmountTitanX > 0) {
            uint256 incentive = _calculateIncentiveFee(claimAmountTitanX,
            ↪  claimIncentiveFeeBPS);
            IERC20 titanX = IERC20(TITANX);

            titanX.safeTransfer(msg.sender, incentive);
            titanX.safeTransfer(STAX_VAULT, claimAmountTitanX - incentive);
-       } else {
-            uint256 incentive =
-                _calculateIncentiveFee(claimAmountEth, claimIncentiveFeeBPS);
-            Address.sendValue(payable(msg.sender), incentive);
        }

+       if (claimAmountEth > 0) {
+           uint256 incentive =
+               _calculateIncentiveFee(claimAmountEth, claimIncentiveFeeBPS);
+           Address.sendValue(payable(msg.sender), incentive);
+       }

        emit Claim();
```

Resolution: Acknowledged

[L-2] `tokenIdsOf` will run out of gas

Context: StaxNFT.sol#L229

Description: Currently, `tokenIdsOf` iterates over all token IDs and returns the ones owned by a user. However, this function will revert if enough tokens (1–2k) are minted, as it performs too many operations and runs out of gas.

Recommendation: If you need to determine which tokens are owned by a user, you can track events and store the data off-chain. Alternatively, you can modify `_mint` to use a mapping `address => uint256[]`, where token IDs are stored per address.

Resolution: Acknowledged

[L-3] Users may exploit the diamond pool during the sale

Context: StaxVault.sol#L340

Description: `StaxVault` features a diamond pool where, if the ratio falls below a 120k multiplier, all generated rewards are sent to the normal reward pool. However, during periods of high minting and burning, causing significant volatility in `totalMultipliers`, some users may exploit this by minting just before the diamond pool is emptied.

Example Scenario:
1. Cycles 1, 2, and 3 experience significant minting.
2. In Cycle 4, many early minters burn their tokens, dropping the ratio below 120k.
3. Alice mints an NFT just in time to capture the diamond pool rewards.
4. Alice profits due to receiving extra rewards in this cycle and facing fewer shareholders.

Recommendation: The likelihood of this occurring is low, and the impact is minor. However, if mitigation is necessary, a simple modification in `_processRewardPool` could be made to restrict this behavior to Cycle 5 and beyond (i.e., after the pre-sale period).

Resolution: Acknowledged

[L-4] `EdenBloomPool` rewards will go to staking instead of the vault

Context: EdenStax/EdenStax.sol#L110

Description: When staking with Eden, users are automatically entered into a raffle if their stake is at least 195_000e18. This raffle distributes a specified amount of TitanX every two weeks to a random winner.

```
if (userShares[msg.sender] >= minSharesToBloom)
 ↪  edenBloomPool.participate(msg.sender);
```

If the winner happens to be the Stax contract, the funds are transferred to it. However, these funds will not be distributed as rewards (sent to the vault). Instead, they will be accounted for in the next staking round. This occurs because the `claim` function transfers only the `claimAmount` to the vault, whereas the `swap` function uses `IERC20(TITANX).balanceOf(address(this))` to swap the funds for Eden and then stake them.

Recommendation: Implementing a solution might be complex, but if necessary, consider adding a function to check if this contract is the winner. If it is, ensure the required TitanX is sent to the vault.

Resolution: Acknowledged

Informational

[I-1] Unused error definitions in StaxBuyBurn contract

Context: StaxBuyBurn.sol#L44, StaxBuyBurn.sol#L46

Description: The contract defines two custom errors `TWAP()` and `ZeroInput()` but never uses them in the implementation. Unused code increases contract size unnecessarily and can lead to confusion during code review and maintenance.

Recommendation: Either remove the unused error definitions or implement their intended error handling logic.

Resolution: Fixed in d7a37d9

[I-2] `purchasesOf` will cost a lot of gas

Context: Stax.sol#L293-L294

Description: The `purchasesOf` function consumes a lot of gas and may revert if the `currentPurchaseId` count exceeds a few thousand. This happens because memory allocation grows exponentially rather than linearly. The function creates arrays sized to `currentPurchaseId`, even though only `end - start` elements are used during iteration.

```
uint64[] memory _purchaseIds = new uint64[](currentPurchaseId);
UserPurchase[] memory _purchases = new UserPurchase[](currentPurchaseId);
```

Recommendation: Reduce the size of the arrays to match the actual number of iterations (`end - start`).

Resolution: Fixed in fbd3ce7