SB SECURITY

Stax

Security Review

Dec 6, 2024

Conducted by:
Blckhv, Lead Security Researcher
Slavcheww, Lead Security Researcher

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	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** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

Stax is an NFT collection and token aimed to be a one stop shop for TitanX stakers. The NFT collection is backed by STAX tokens, and can be burned at any time to redeem STAX. By holding the NFT users are eligible to rewards from the TitanX projects' staking.

Overview

Project	Stax
Repository	Private
Commit Hash	b28964ab4d371bf37c53392a479b05d64d5b32b1
Resolution	45f0917ecfa9966baf2ca1a8a4a4aa99307876b7
Timeline	November 26 - December 6, 2024

Scope

/contracts

Issues Found

Critical Risk	0
High Risk	1
Medium Risk	11
Low/Info Risk	23

5. Findings

5.1. High severity

5.1.1. rewardPool is not lowered upon distribution

Severity: High Risk

Description: HyperStax uses rewardPool because the rewards are in the form of Hyper and the contract must distinguish the rewards from the other balance. But when distributeRewards() is called, there is a maxDistributionAmount in Hyper which is 100B, and when the rewards are more, only 100B is exchanged and transferred to BuyAndBurn, but the rewardPool is not decremented, and the next call will assume that there are more rewards and failed.

```
function distributeRewards(uint256 minAmountOut, uint256 deadline) external originCheck {
    if (!whitelisted[msg.sender]) revert Prohibited();
    (uint256 time, uint256 amount) = getNextDistributionInfo();
    if (time > block.timestamp) revert Cooldown();
    if (amount == 0) revert InsufficientBalance();
    lastDistributionTime = block.timestamp;
    uint256 incentive = _calculateIncentiveFee(amount, distributeIncentiveFeeBPS);
    _swapHyperToElement(amount - incentive, minAmountOut, deadline);
    IERC20(HYPER).safeTransfer(msg.sender, incentive);
    emit Distribution();
}
```

```
function getNextDistributionInfo() public view returns (uint256 time, uint256 amount) {
    amount = rewardPool > maxDistributionAmount ? maxDistributionAmount : rewardPool;
    time = lastDistributionTime + distributionInterval;
}
```

Also in getNextStakeInfo(), since the rewardPool is only incremented when rewards are swapped, the rewardPool will continue to decrease the stake balance, and stake() will either revert or stake less Hyper.

Recommendation: In distributeRewards(), decrement the rewardPool by the amount that was used.

Resolution: Fixed

5.2. Medium severity

5.2.1. `BlazeDiamondHand` should have setter

Severity: Medium Risk

Description: BlazeDiamondHand must have a setter because in `BlazeStaking` it can also be changed with a setter.

```
BlazeStaking.sol

function setLastDistributionAddress(address __lastDistributionAddress) external onlyOwner {
    require(__lastDistributionAddress != address(0), "blazeStaking:last distribution address can not be
zero");
    _lastDistributionAddress = __lastDistributionAddress;
}
```

Recommendation: Make a setter for the `BLAZE_DIAMOND_HAND` in `BlazeStax.sol`.

Resolution: Fixed

5.2.2. After reward tokens stop entering the staking contracts, TitanX will be locked in Stax

Severity: Medium Risk

Description: After rewards tokens stop entering the underlying protocols, tokens received after stakes mature will be locked in the Staking contracts.

- TitanX - when no more miners are being started, `ETH` payouts will dry out - `TitanX` will be locked in the `TitanXStax`.
- Helios - when no more miners are being started, both `ETH` and `TitanX` payouts will dry out - `TitanX` will be locked in the `TitanXStax`.
- Blaze - when tokens won from auction, don't outperform the ETH required for participation - `Blaze` will be locked in `BlazeXStax`.
- Flux - when the last 2888th day of Auction ends, no more TitanX will be allocated for Staking rewards - `Flux` will be locked in `FluxStax`.
- Hyper - when no more miners are being started Hyper payouts will stop - `Hyper` that is not part of `rewardPool` will be locked in `HyperStax`.

Recommendation: Add a mechanism to retrieve the locked funds after no more rewards are being issued.

Resolution: Acknowledged

5.2.3. Incentive will be 0 if only Eth rewards are accumulated

Severity: Medium Risk

Description: In `HeliosStax::claim()`, if the accumulated rewards are only in the form of ETH when the function is called, there will be no incentive for the caller.

```
function claim() external originCheck {
    uint256 claimAmountTitanX;
    uint256 claimAmountEth;
    for (uint256 i = 0; i < numInstances; i++) {
        address payable instance = payable(instances[i]);
        (uint256 claimedTitanX, uint256 claimedEth) = HeliosStaxInstance((instance)).claim();
        claimAmountTitanX += claimedTitanX;
        claimAmountEth += claimedEth;
    }
    if (claimAmountTitanX == 0 && claimAmountEth == 0) revert NothingToClaim();

    uint256 incentive = _calculateIncentiveFee(claimAmountTitanX, claimIncentiveFeeBPS);
    IERC20 titanX = IERC20(TITANX);
    titanX.safeTransfer(msg.sender, incentive);
    titanX.safeTransfer(STAX_VAULT, claimAmountTitanX - incentive);
    emit Claim();
}
```

Recommendation: Send an incentive from the claimed rewards from the reward that is not 0.

Resolution: Fixed

5.2.4. Precision loss in rewards per multiplier calculations

Severity: Medium Risk

Description: The current implementation of rewards per multiplier is not immune to loss of precision, some percentage of the rewards will be lost for the current cycle. The loss can be up to `totalMultipliers - 1` for the current cycle.

```
uint256 perMultiplier = rewardsAmount / totalMultipliers;
tokensPerMultiplier[cycle] = tokensPerMultiplier[cycle - 1] + perMultiplier;
```

Recommendation: Scale the `rewardsAmount` so there is no precision loss.

Resolution: Acknowledged

5.2.5. If liquidity providers don't close the titanX/e280 arbitrage, stax/e280 ratio will be broken

Severity: Medium Risk

Description: 150B TitanX will be swapped for as much as possible E280, but at the time of writing in the TitanX/E280 pair there is 149.23B E280.

In order to exchange the TitanX, receiving E280 at the uninflated market price we will need ~185B E280 (TitanX/E280 ratio 1:1.2346), meaning that if there are no arbitrages b/n each `purchaseTokenForLP` we will receive way less Element than intended resulting in a completely different `E280/Stax` LP ratio when `deployStaxLP` is called.

Recommendation: It was assured by the protocol team that there are arbitrage bots, who will take the opportunity and provide the missing `E280` once the purchases begin.

Resolution: Acknowledged

5.2.6. Blaze diamond hand rewards can be lost if tickets are not up to date

Severity: Medium Risk

Description: BlazeDiamondHand rewards may be lost if `participate()` is not called.

If there was a stake before the 365th day (the day until which `diamondHand` rewards are awarded), but after that stake `participate()` is not called, prizes equivalent to that stake will not be given and will not be left for the next cycle (this bet simply won't count towards prizes).

Although this is more of a Blaze contract infrastructure, you should be aware of this.

Recommendation: Make sure that after the 365th day of blazeStaking, `participate()` will be called by the Stax team or someone else.

Resolution: Acknowledged

5.2.7. Use swapExactTokensForTokensSupportingFeeOnTransferTokens when swapping to/from E280

Severity: Medium Risk

Description: Since the E280 token has a transfer tax, `Router.swapExactTokensForTokensSupportingFeeOnTransferTokens()` must be used, which only swaps the E280 balance after the tax received in the router, otherwise the swap function may revert as it will try to swap more tokens.

Recommendation: Change all swaps that have E280 as `token0/token1` to use `swapExactTokensForTokensSupportingFeeOnTransferTokens`.

Resolution: Acknowledged, swaps that do not use feeOnTransferSwap are from/to StaxBnB contract and it will be excluded from E280 fees

5.2.8. Staking contracts stakes should be limited to 100B per stake

Severity: Medium Risk

Description: Both `TitanX`, `Helios` and `Hyper` stakes have their max shares bonus capped at 100B stake, but staking contracts **always** stake the entire token balance whenever `stake` is called.

```
function getNextStakeInfo() public view returns (uint256 time, uint256 amount) {
    amount = IERC20(TITANX).balanceOf(address(this));
    time = lastStakeTime + stakeInterval;
}
```

Recommendation: Add max stake amount, which is 100B, if the token balance is above it, stake only 100B, otherwise entire balance.

Resolution: Acknowledged

5.2.9. Last stake reward lost if `triggerPayouts` not called before unstake

Severity: Medium Risk

Description: If `endStakeAfterMaturity` is executed before `triggerPayout` for the last cycle, rewards for at least one reward cycle (depending on when the last time `triggerPayouts` is called) are lost. They cannot be claimed after that, since Stax shares of that stake id in `TitanX`, `Helios`, `Hyper` and `Blaze` are marked as inactive, here is a snippet from the `TitanX` code but the other's logic is the same:

```
function _endStake(
    address user,
    uint256 id,
    uint256 day,
    StakeAction action,
    StakeAction payOther,
    PayoutTriggered isPayoutTriggered
) internal returns (uint256 titan) {
    uint256 globalStakeId = s_addressSIdToGlobalStakeId[user][id];
    if (globalStakeId == 0) revert TitanX_NoStakeExists();

    UserStakeInfo memory userStakeInfo = s_globalStakeIdToStakeInfo[globalStakeId];
    if (userStakeInfo.status == StakeStatus.ENDED) revert TitanX_StakeHasEnded();
    if (userStakeInfo.status == StakeStatus.BURNED) revert TitanX_StakeHasBurned();
    //end stake for others requires matured stake to prevent EES for others
    if (payOther == StakeAction.END_OTHER && block.timestamp < userStakeInfo.maturityTs)
        revert TitanX_StakeNotMatured();

    //update shares changes
    uint256 shares = userStakeInfo.shares;
    _updateSharesStats(user, shares, userStakeInfo.titanAmount, day, isPayoutTriggered, action);

    if (action == StakeAction.END) {
        ++s_globalStakeEnd;
        s_globalStakeIdToStakeInfo[globalStakeId].status = StakeStatus.ENDED;
    } else if (action == StakeAction.BURN) {
        ++s_globalStakeBurn;
        s_globalStakeIdToStakeInfo[globalStakeId].status = StakeStatus.BURNED;
    }

    titan = _calculatePrinciple(user, globalStakeId, userStakeInfo, action);
}
```

```
function _updateSharesStats(
    address user,
    uint256 shares,
    uint256 amount,
    uint256 day,
    PayoutTriggered isPayoutTriggered,
    StakeAction action
) private returns (uint256 isFirstShares) {
    //Get previous active shares to calculate new shares change
    uint256 index = s_userSharesIndex[user];
    uint256 previousShares = s_addressIdToActiveShares[user][index].activeShares;

    if (action == StakeAction.START) {
        //return 1 if this is a new wallet address
        //this is used to initialize last claim index to the latest cycle index
        if (index == 0) isFirstShares = 1;

        s_addressIdToActiveShares[user][++index].activeShares = previousShares + shares;
        s_globalShares += shares;
        s_globalTitanStaked += amount;
    } else {
        s_addressIdToActiveShares[user][++index].activeShares = previousShares - shares;
        s_globalExpiredShares += shares;
        s_globalTitanStaked -= amount;
    }

    //If global payout hasn't triggered, use current contract day to eligible for payout
    //If global payout has triggered, then start with next contract day as it's no longer eligible to claim
latest payout
    s_addressIdToActiveShares[user][index].day = uint128(
        isPayoutTriggered == PayoutTriggered.NO ? day : day + 1
    );

    s_userSharesIndex[user] = index;
}
```

That happens because the last reward cycle is not finalized yet and the global share index is not updated at the time of unstake stake's index == global index, although a new cycle has begun.

The mechanics are the same as in `claim`, but here we can't claim later because of the above-mentioned equality.

`TitanXStax`, `HeliosStax`, `HyperStax` and `BlazeStax` are vulnerable to not receiving their last payouts, only Flux isn't because it always checks for pending rewards.

Recommendation: Make sure to call `triggerPayouts` in `endStakeAfterMaturity` of all 3 staking contracts, and `distributeFeeRewardsForAll` in `BlazeStax`, also make sure to redirect the incentive to the caller.

Resolution: Fixed

5.3. Low/Info severity

5.3.1. Low and informational issues

Severity: Low Risk

Description:

1. `_checkPercentageArray()` doesn't check for `percentage = 0`.
2. Add `twap` checks to `WETH/TITANX` swaps.
3. In `_checkPoolValidity()`, use `pair.skim()` instead of `pair.sync()`, if there are tokens sent to the contract but not synced, it will skip adding liquidity manually and act normally.
4. In `purchaseWithTitanX()`, use a ternary operator for the receiver as in `purchaseWithETH()`.
5. Fix `claimRewards()` NatSpec's param to be for single tokenId.
6. In `stakeReachedMaturity()` the check for `maturityTs` should be `block.timestamp >= stakes[i].stakeInfo.maturityTs`. This is because the stake matures at `maturityTs`.
7. Use `onlyToken` on `StaxVault::handleStartPresale()`.
8. `_fixPool()` is useless because in this case the liquidity providing is done manually and the ratio cannot be saved if someone had sent E280 to the pool. Remove `_fixPool()` while you're sending all the Stax and e280 in `_deployStaxLiqudityPool`.
9. `mintCycle` in `_setNftMintData` and `_packMintData` should be `uint32` instead of `uint64`.
10. `maxSwapValue` in `StaxVault.sol` is not in 18 decimals which will only swap 50B wei of TitanX to X28 instead of 50B TitanX.
11. Add a view function in `StaxStaking` contracts that will return the claimable rewards (if any) for all instances. This will help prevent claim calls if there are no rewards, and this information can be used on the FE.
12. Wrong number of NFT tiers in `getTotalNftsPerTiers()`. NFT tiers are 6, not 24. Change the `total` array to only 6 elements.
13. Remove `_getNftTier()` from `StaxVault.sol`, it is not used.
14. The `claim()` function in the `StaxStaking` will revert when many instances are deployed. Add `start` and `end` index to `claim()` functions.
15. In `addNewStaxContract()`, add a check ensuring that `staxContract` is a real `StaxStaking` contract by checking one of its properties.
16. `_deployStaxLiquidityPool()` uses `block.timestamp` as `deadline` for `addLiquidity`. Pass custom `deadline` instead.

17. `_tokenTargets` in `StaxBank` functions can be used as `memory` and then reassigned like the other mappings.

18. Add setter for `minDistributionAmount`.

19. Change the error in `enableNewDistributionToken()/addNewStaxContract` from `DuplicateToken` to `DuplicateTarget`.

20. `DistributionChanged` event is not used, either use it or remove it.

21. `_baseURI` in `StaxNFT` shadows existing variable from `ERC721A`.

22. `StaxBuyBurn::buyAndBurn` does 2 swaps when `e280` balance is below `capPerSwapE280`, in the cases when these 2 swaps are performed higher incentive must be given since the gas fees will be higher and there is a chance `Stax` burns not to happen in a timely manner. You can track the count of swaps manually and give double incentive to the caller.

23. In staking contracts with instances, there is a missing is `isAtMaxStakes` in the `stake` functions. Currently, the execution will be reverting in the target protocol and not in Stax. In order to fix that, `isAtMaxStakes` have to be called in `TitanXStax`, `HyperStax` and `HeliosStax`

Resolution: Fixed - 1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17, 18, 19, 20, 21, 22