Element369

Security Review

Nov 15, 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

Element 369 (E369) is an expansion pack NFT collection on the Element 280 protocol. By holding the E369 NFT, you earn E280, Inferno, and Flux tokens via perpetual FLUX auction participation without having to add additional investment capital.

Overview

Project	Element369
Repository	Private
Commit Hash	06d348a291bd31f47e390ee3c7b1aa3bc1d6c182
Resolution	26e82e5aa8fa3f7bcce9131e9fd2b61f492d1a63
Timeline	November 11 - November 15, 2024

Scope

Element369HolderVault.sol
Element369NFT.sol
ElementBuyBurnV2.sol
FluxHub.sol
DevDistribute.sol

Issues Found

Critical Risk	0
High Risk	2
Medium Risk	2
Low/Info Risk	0

5. Findings

5.1. High severity

5.1.1. 777 Rewards for NFTs entered at cycle 34th can be blocked by anyone

Severity: High Risk

Description: All NFTs that are minted before the start of 35th cycle start (34th cycle is included) must be added to the 777 rewards. But due to wrong check in `batchBurn`, anyone can mint an NFT at the beginning of 34th cycle and then burn it, which will set the `multiplerPool` up to that moment for the 777 rewards and all NFTs minted in the 34th cycle will not be added because when 35th cycle starts and `Element369HolderVault.updateStoredMultipliers()` is called, multiplierPool for this 777 cycle (1st) will already be updated by the attacker and the function will return.

This happens because the `if` check will hit even when `currentCycle` is 34th because of the >=.

```
function batchBurn(uint256[] calldata tokenIds) external {
    if (tokenIds.length == 0) revert ZeroInput();
    uint32 currentCycle = getCurrentE369Cycle();
    uint32 currentCycle777 = getCurrentCycle777();
    uint32 startCycleId = getStartCycleForCycle777(currentCycle777);
    uint32 endCycleId = getEndCycleForCycle777(currentCycle777);
    if (currentCycle >= startCycleId && currentCycle < endCycleId && currentCycle777 <
MAX_777_CYCLES_NUMBER) {
        uint256 totalApplicableMultipliers;
        for (uint256 i = 0; i < tokenIds.length; i++) {
            uint256 tokenId = tokenIds[i];
            if (ownerOf(tokenId) != msg.sender) revert Unauthorized();
            _burn(tokenId);
            (uint16 multiplier, uint64 mintCycle) = _updateBurnCycleConditional(tokenId, currentCycle,
msg.sender);
            if (mintCycle <= startCycleId) totalApplicableMultipliers += multiplier;
            multiplierPool -= multiplier;
        }
        IE369HolderVault(HolderVault).updateStoredMultipliersOnBurn(currentCycle777, multiplierPool,
totalApplicableMultipliers);
    } else {
        for (uint256 i = 0; i < tokenIds.length; i++) {
            uint256 tokenId = tokenIds[i];
            if (ownerOf(tokenId) != msg.sender) revert Unauthorized();
            _burn(tokenId);
            multiplierPool -= _updateBurnCycle(tokenId, currentCycle, msg.sender);
        }
    }
}
```

```
function updateStoredMultipliers(uint32 cycleId, uint256 totalMultipliers) external {
    if (msg.sender != E369_NFT) revert Unauthorized();
    if (cycles777[cycleId].multiplierPool != 0) return;
    cycles777[cycleId].multiplierPool = totalMultipliers;
}
```

Recommendation: Remove the `=` from the if check.

Resolution: Fixed

5.1.2. Users rewards will be lost forever because claim rewards updating the last updated to the current cycle

Severity: High Risk

Description: Users rewards will be lost forever when he claims in a cycle that is not updated. This is because in `claimRewards()` when the NFT is not burned it gets the **`current cycle`** and loop up to this one in `_processCycles()` and because the cycle will not be updated it will be skipped from rewards, but the **`current cycle`** is returned from the function and is assigned to `nftLastClaim` and then when those cycles that have not been updated are updated the user will lost his rewards for them as the next time `claimRewards()` is called it will collect the rewards starting from `nftLastClaim` which was set to the cycle that wasn't updated.

```
/// @notice Claims the accumulated rewards for a batch of NFT tokens.
/// @param tokenIds An array of token IDs for which rewards are being claimed.
function claimRewards(uint256[] calldata tokenIds) external {
    if (tokenIds.length == 0) revert ZeroInput();
    uint32 currentCycle = getCurrentE369Cycle();
    uint256[] memory nftData = IElement369NFT(E369_NFT).batchGetPackedTokenData(tokenIds, msg.sender);
    uint256 totalInferno;
    uint256 totalFlux;
    uint256 totalE280;
    for (uint256 i = 0; i < tokenIds.length; i++) {
        uint256 tokenId = tokenIds[i];
        uint256 tokenData = nftData[i];
        uint32 nftMintCycle = _getMintCycle(tokenData);
        uint32 nftBurnCycle = _getBurnCycle(tokenData);
        uint32 lastCycle = nftBurnCycle > 0 ? nftBurnCycle : currentCycle; <--------------
        uint16 nftMultiplier = _getMultiplier(tokenData);
        uint32 nftLastClaimed = nftLastClaim[tokenId];
        (uint32 endCycle, uint256 infernoPool, uint256 fluxPool, uint256 e280Pool) =
            _processCycles(nftMintCycle, nftMultiplier, nftLastClaimed, lastCycle);
        totalInferno += infernoPool;
        totalFlux += fluxPool;
        totalE280 += e280Pool;
        nftLastClaim[tokenId] = endCycle; <------------------
    }
    _processTokenRewardsPayout(INFERNO, msg.sender, totalInferno);
    _processTokenRewardsPayout(FLUX, msg.sender, totalFlux);
    _processTokenRewardsPayout(E280, msg.sender, totalE280);
}
```

```
function _processCycles(uint32 nftMintCycle, uint16 nftMultiplier, uint32 lastClaim, uint32 maxCycle)
    internal
    view
    returns (uint32 endCycle, uint256 infernoPool, uint256 fluxPool, uint256 e280Pool)
{
    uint32 startCycle = lastClaim == 0 ? nftMintCycle : lastClaim;
    if (startCycle == maxCycle) revert NoCyclesAvailable();
    endCycle = _applyMaxCycleProtection(startCycle, maxCycle);  <-------------------------------------
    …
}
```

Recommendation: Store the last updated cycle and then in `claimRewards()` pass the last updated cycle + 1 for `lastCycle`.

Resolution: Fixed

5.2. Medium severity

5.2.1. Multiplier for 5th 777 rewards cycle will be updated even the protocol have only 4 cycles

Severity: Medium Risk

Description: Wrong check in `updateMultipliersIfNecessary()` will cause NFTs `multiplierPool` to be set for 5th 777 rewards cycle, while the protocol should only have 4 777 rewards cycles.

```
function updateMultipliersIfNecessary() public {
    uint32 currentCycle = getCurrentE369Cycle();
    uint32 currentCycle777 = getCurrentCycle777();
    if (currentCycle777 > MAX_777_CYCLES_NUMBER) return;      <-----------------------------------
    if (!is777MultipliersSet[currentCycle777] && currentCycle > getStartCycleForCycle777(currentCycle777))
{
        IE369HolderVault(HolderVault).updateStoredMultipliers(currentCycle777, multiplierPool);
        is777MultipliersSet[currentCycle777] = true;
    }
}
```

The if check should not allow `currentCycle777` to be 4, this is because the 1st cycle is actually 0 and the last 4th will be numbered 3.

This does not result in a loss of rewards for the remaining cycles, but will update the multiplier for `cycles777[4]` which can then be accessed in `_processTokenIdCycles777()`, as `claim777Rewards()` uses a slightly different `getCurrentCycle777()` for `Element369HolderVault` purposes.

Recommendation: Change the if check in `updateMultipliersIfNecessary` to `currentCycle777 ≥ MAX_777_CYCLES_NUMBER`.

Resolution: Fixed

5.2.2. `_applyMaxCycleProtection` wrongly removes 1 cycle

Severity: Medium Risk

Description: When users claim their rewards, there is max cycles that can be claimed, which is set to 100 (`MAX_CYCLES_PER_CLAIM`). The function that limits this adds 100 to the starting cycle from which the rewards will be claimed and removes 1, but since the result from this is used as `end` of a for loop and it is excluded, the actual cycles that will be preceded will be 99 not 100.

```
function _applyMaxCycleProtection(uint32 startCycle, uint32 currentCycle) internal pure returns (uint32) {
    uint32 maxEndCycle = startCycle + MAX_CYCLES_PER_CLAIM - 1;
    return currentCycle < maxEndCycle ? currentCycle : maxEndCycle;
}
```

```
function _processCycles(uint32 nftMintCycle, uint16 nftMultiplier, uint32 lastClaim, uint32 maxCycle)
    internal
    view
    returns (uint32 endCycle, uint256 infernoPool, uint256 fluxPool, uint256 e280Pool)
{
    uint32 startCycle = lastClaim == 0 ? nftMintCycle : lastClaim;
    if (startCycle == maxCycle) revert NoCyclesAvailable();
    endCycle = _applyMaxCycleProtection(startCycle, maxCycle);
    unchecked {
        for (uint256 i = startCycle; i < endCycle; i++) { <---------------- endCycle is excluded
            Cycle memory cycle = cycles[i];
            if (!cycle.initialized) continue;
            infernoPool += cycle.infernoPerMulitplier;
            fluxPool += cycle.fluxPerMultiplier;
            e280Pool += cycle.e280PerMultiplier;
        }
    }
    infernoPool = (infernoPool * nftMultiplier) / 1e18;
    fluxPool = (fluxPool * nftMultiplier) / 1e18;
    e280Pool = (e280Pool * nftMultiplier) / 1e18;
}
```

Recommendation: Do not subtract 1 in `_applyMaxCycleProtection()`.

Resolution: Fixed