Baton Launchpad Audit Report

Prepared by bytes032

Contents

1 About bytes032

bytes032 is an independent smart contract security researcher.

His knack for identifying smart contract security vulnerabilities in a range of protocols is more than a skill; it's a passion. Committed to enhancing the blockchain ecosystem, he invests his time and energy into thorough security research and reviews. Want to know more or collaborate? bytes's always up for a chat about all things security.

Feel free to reach out on X.

2 Protocol Summary

Baton Launchpad offers a suite of features that can be configured and tuned by creators to maximize the fairness and amount of liquidity for their NFT. These features include refunds, staggered mints, locked liquidity, yield farming, and vesting. The following sections will explain how each of these features works. Each of the following features is optional and can be configured by the creator.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of {X} according to the specific commit. Any modifications to the code will require a new security review.

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 loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Overview

Project	Baton Launchpad
Repository	baton-launchpad
Commit	64444c759956...
Date	August 2023

Issues Found

Severity	Count
Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	1
Informational	1
Gas Optimizations	0
Total Issues	**5**

Summary of Findings

Title	Status
[M-1] Unbounded royalty rate in NFT Creation	Resolved
[M-2] Inconsistent Fee Rate Limitation	Resolved
[M-3] Refund Miscalculation Due to Inclusion of Free NFTs	Acknowledged
[L-1] Possibility of griefing when locking LP	Acknowledged
[I-1] Redundant event emission	Resolved

5 Findings

5.1 Medium Risk

[M-1] Unbounded royalty rate in NFT Creation

Context: BatonLaunchpad.sol

Impact: If a user sets the royalty rate to 100% when creating an NFT, the original seller will not receive any proceeds when the NFT is sold in the secondary market.

Description: The code snippet provided shows the mechanism for creating an NFT, specifically with a focus on the royalty rate. The vulnerability arises because there is no upper limit check on the royalty rate. As per the description, a user can set the royalty rate to 100%, implying that when the NFT is subsequently sold, the entire sale price would be taken as royalty, leaving the seller with no proceeds from the sale.

Recommendation: Implement a check to ensure the royalty rate is within acceptable limits, ideally less than 100%. This would ensure that the original seller receives at least a portion of the sale proceeds when the NFT is resold in the secondary market.

[M-2] Inconsistent Fee Rate Limitation

Context: BatonLaunchpad.sol

Impact

This inconsistency allows an owner to bypass the fee rate limitation set in the setFeeRate function by setting a fee rate higher than 10% at the time of contract deployment.

Description

In the provided setFeeRate function from the BatonLaunchpad.sol file, there's an explicit check to ensure that the fee rate does not exceed 10%.

When setting a fee rate, the owner is limited with a max fee rate of 10%.

However, this check is not present in the contract's constructor. Thus, when deploying the contract, the owner can set a fee rate higher than the intended 10% limit, effectively circumventing the limitation imposed by the setFeeRate function.

Recommendation

Ensure consistency in fee rate limitations by implementing the same fee rate check in the contract's constructor. This will prevent the owner from setting a fee rate above the intended 10% limit during contract deployment.

[M-3] Refund Miscalculation Due to Inclusion of Free NFTs

Context: Nft.sol

Impact: The current implementation of the refund mechanism does not differentiate between paid and free NFT mints. As a result, users who have minted free NFTs could, under certain cirumstances receive a lower refund amount than what they are actually entitled to, causing potential monetary loss.

Description: In the provided logic, when an NFT is minted and refunds are enabled, the totalMinted counter for an address gets incremented regardless of whether the NFT is free or paid.

When a refund is requested, the refund amount is calculated based on the number of NFTs held and the totalMinted counter, without accounting for the actual value of every individual NFT, but based on the total amount.

However, let's assume there are multiple categories and one of them is free, let's say:

Amelie mints 3 free NFT's from `category[0]` and 5 paid NFT's from `categories[1]`. If, when the refund is available Amelie requests a refund - she should be entitled her 5 ETH back.

Let's say Amelie gives away 2 of her free NFT's (`category[0]`).

Please, note that even though she has only **"five paid NFT's"**, her `totalMinted = 8`, because the function doesn't make a difference between free or paid mints.

Now, if Amelie requests a refund and holds on 6 of her initially 8 NFT's, she would only be able to get back 3.7e18 (6 * 5e18 / 8) and the rest of the funds are lost unless she's able to reclaim any of the other NFT's.

Recommendation

Adjust the logic so that the totalMinted counter only increments when a paid NFT is minted. Specifically, add a conditional check to see if the category.price of the minted NFT is greater than 0 before incrementing the totalMinted counter. This will ensure that only paid mints contribute to the refund calculation, safeguarding users from potential monetary loss.

5.2 Low Risk

[L-1] Possibility of griefing when locking LP

Context: Nft.sol

Impact: Malicious actors can exploit the current lockLp function to consistently front-run genuine locking transactions. While these attackers may not gain any direct benefits, their actions can cause monetary loss for honest users in the form of wasted gas and disrupt the protocol's operations by delaying the locking process.

Description

In the provided lockLp function:

The process allows for incremental increases in the lockedLpSupply based on the amount parameter. If a malicious actor, like Noemi in the given example, continually front-runs genuine locking transactions with small amounts (e.g., 1 token), the honest user's transaction (like Amelie's) will be reverted due to the "InsufficientLpAmount" condition. Consequently, the honest user will lose the gas paid for the transaction, and the locking process will be delayed.

Recommendation

To mitigate this vulnerability, consider implementing a batching mechanism that processes multiple lock requests in a single transaction, reducing the opportunities for front-running. Additionally, consider using flashbots, which would allow the transaction to be processed privately and avoid being front-run altogether. Adopting such strategies will ensure the protocol's resilience against malicious front-running tactics.

5.3 Informational

[I-1] Redundant event emission

Context: Nft.sol

Description

In the provided refund function, if a user calls it with an empty array (tokenIds.length == 0), the function will still proceed without any side effects until it emits a refund event at the end. This can lead to false indications that a refund took place when, in reality, no tokens were refunded.

This behavior is misleading and could cause confusion or misinterpretation of contract events.