Baton Launchpad Security Review

Pashov Audit Group

Conducted by: pashov

August 27th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Baton Launchpad** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Baton Launchpad

Baton Launchpad is an NFT launchpad protocol. It has multiple features that allow for NFT creators to maximize the liquidity of their tokens. You can configure refunds, staggered mints, yield farming, vesting and locking liquidity.

If by a given time the tokens are not minted out, users can issue a refund by burning their NFT. Mints can also be configured to be in different categories that might require a whitelist or a different mint price. Another thing is that the NFTs allocated to team, investors etc. can be vested over time and the users can be certain that they won't be insta-dumped on the secondary market after mint.

Observations

Even if an NFT doesn't mint out and people get a refund by burning their tokens, vesting is still possible for token creators.

The `maxMintSupply` will be less than the `totalSupply` if there are tokens to vest and the collection is minted out - vested tokens do not count in `maxMintSupply` as well as ones that are used for yield farm or as locked LP.

External dependencies are baton-contracts and Caviar - both are audited.

Privileged Roles & Actors

- Baton Launchpad owner - can configure the fee rate for NFT mints and the `nftImplementation` contract to be cloned, as well as execute a migration of Locked LP tokens in `Nft`
- Nft owner - can withdraw the mint payouts and also set the target contract for a potential locked LP migration
- Vesting receiver - can call `vest` to claim vested NFT tokens after a certain time threshold
- Whitelisted Minter - can mint NFTs from a category that requires whitelisting by submitting a `proof`
- Minter - can mint NFTs from categories that do not require whitelisting

Minters can also refund their ETH if mint hasn't completed by a certain time threshold and refunding is enabled.

Anybody can call `lockLp` and `seedYieldFarm` to mint tokens to either a `Pair` contract or a `BatonFarm` one.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. 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

6. Security Assessment Summary

review commit hash **- d278f930ef3e358f61fa5c42a73d197059aa2dad**

fixes review commit hash **- 709e1df99d433266554dc5a551c4ab325bcdc2cb**

Scope

The following smart contracts were in scope of the audit:

- `BatonLaunchpad`
- `Nft`

7. Executive Summary

Over the course of the security review, pashov engaged with Baton Launchpad to review Baton Launchpad. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Baton Launchpad
Date	August 27th, 2023

Findings Count

Severity	**Amount**
Critical	1
High	1
Medium	3
Low	2
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Protocol fees from NFT mints can't be claimed in BatonLaunchpad	Critical	Resolved
[H-01]	Missing user input validation can lead to stuck funds	High	Resolved
[M-01]	It's not possible to execute a rewards migration of a BatonFarm	Medium	Resolved
[M-02]	Possible front-running griefing attack on NFT creations	Medium	Resolved
[M-03]	Centralization vulnerabilities are present in the protocol	Medium	Resolved
[L-01]	The payable methods in Nft can result in stuck ETH	Low	Resolved
[L-02]	The refund mechanism can be used by accounts with allowances	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Protocol fees from NFT mints can't be claimed in `BatonLaunchpad`

Severity

Impact: High, as it results in a loss of value for the protocol

Likelihood: High, as it certain to happen

Description

In `Nft::mint` the `msg.value` expected is the price of an NFT multiplied by the amount of NFTs to mint plus a protocol fee. This protocol fee is sent to the `BatonLaunchpad` contract in the end of the `mint` method like this:

```
if (protocolFee != 0) {
    address(batonLaunchpad).safeTransferETH(protocolFee);
}
```

`BatonLaunchpad` defines a `receive` method that is marked as `payable`, which is correct. The problem is that in `BatonLaunchpad` there is no way to get the ETH balance out of it - it can't be spent in any way possible, leaving it stuck in the contract forever.

Recommendations

In `BatonLaunchpad` add a method by which the `owner` of the contract can withdraw its ETH balance.

8.2. High Findings

[H-01] Missing user input validation can lead to stuck funds

Severity

Impact: High, as all mint fees can be stuck forever

Likelihood: Medium, as users can easily misconfigure inputs

Description

There are multiple insufficiencies in the input validation of the arguments of the `initialize` method in `Nft`:

1. The sum of the `supply` of all `categories_` can be less than the `maxMintSupply_` - this would lead to the mint never completing, which results in all of the ETH in the `Nft` contract coming from mints so far being stuck in it forever
2. The `duration` of the `vestingParams_` should have a lower and upper bound as for example a too big of a duration can mean vesting can never complete or a division rounding error
3. The `mintEndTimestamp` of `refundParams_` should not be too further away in the future otherwise refund & vesting mechanisms would never work, and if it is too close then the mint mechanism won't work.

Recommendations

Add a validation that the sum of all categories' supply is more than or equal to the `maxMintSupply`. Also add sensible upper and lower bounds for both `duration` for the vesting mechanism and `mintEndTimestamp` for the refund mechanism.

8.3. Medium Findings

[M-01] It's not possible to execute a rewards migration of a `BatonFarm`

Severity

Impact: High, as it can lead to stuck rewards

Likelihood: Low, as it is not likely that a migration is needed

Description

The `BatonFarm` contract which is an external dependency of the `Nft` contract (a `BatonFarm` is deployed in `seedYieldFarm`) has a migration mechanism to move the unearned rewards to a new contract. This functionality is currently blocked, because it depends on a call from the `BatonFarm` owner (the `Nft` contract in this case) to the `initiateMigration` method of `BatonFarm`. Since such a call is not possible as there is no code for it, migrations are currently impossible in the system. This means that if there are rewards left in a `BatonFarm` contract deployed by some `Nft` contract, they will be stuck there forever.

Recommendations

Add a way for the `Nft` admin to execute an `initiateMigration` call.

[M-02] Possible front-running griefing attack on NFT creations

Severity

Impact: Medium, as it results in a temporary DoS for users of the protocol

Likelihood: Medium, as it is easy to execute but attacker doesn't have much incentive to do it

Description

The `create` method in `BatonLaunchpad` calls the `cloneDeterministically` method from `LibClone` that uses the `create2` opcode. The `create` method also has a `salt` parameter that is passed to the `cloneDeterministically` call. A malicious actor can front-run every call to `create` and use the same `salt` argument. This will result in reverts of all user transactions, as there is already a contract at the address that `create2` tries to deploy to.

Recommendations

Adding `msg.sender` to the `salt` argument passed to `cloneDeterministically` will resolve this issue.

[M-03] Centralization vulnerabilities are present in the protocol

Severity

Impact: High, as it can lead to a rug pull

Likelihood: Low, as it requires a compromised or a malicious owner

Description

The `owner` of `BatonLaunchpad` has total control of the `nftImplementation` and `feeRate` storage variable values in the contract. This opens up some attack vectors:

1. The `owner` of `BatonLaunchpad` can front-run a `create` call to change the `nftImplementation` contract to one that also has a method with which he can withdraw the mint fees from it, resulting in a "rug pull"
2. The `owner` of `BatonLaunchpad` can change the fee to a much higher value, either forcing the `Nft` minters to pay a huge fee or just to make them not want to mint any tokens.
3. The `owner` of the `Caviar` dependency can call `close` on the `Pair` contract, meaning that the `nftAdd` call in `lockLp` and the `wrap` call in `seedYieldFarm` would revert. This can mean that the locking of LP and the seeding of the

yield farm can never complete, meaning the `owner` of the `Nft` contract can never call `withdraw`, leading to stuck ETH in the contract.

Recommendations

Make the `nftImplementation` method callable only once, so the value can't be updated after initially set. For the `feeRate` add a `MAX_FEE_RATE` constant value and check that the new value is less than or equal to it. For the `Caviar` dependency issue you can call it with `try-catch` and just complete the locking of LP or seeding of the yield farm if the call throws an error.

8.4. Low Findings

[L-01] The `payable` methods in `Nft` can result in stuck ETH

Multiple methods in `ERC721AUpgradeable` (for example the overriden `transferFrom`) have the `payable` keyword, which means they can accept ETH. While this is a gas optimization, it can result in ETH getting stuck in the `Nft` contract, as it inherits `ERC721AUpgradeable`. You can override `payable` methods and revert on `msg.value != 0` to protect from this problem.

[L-02] The `refund` mechanism can be used by accounts with allowances

The `refund` method calls `_burn` which would allow burning a token if you have allowances for it. While this is a highly unlikely scenario to occur as it also requires the `msg.sender` to have `totalMinted` and `availableRefund` values in the `_accounts` mapping, it is still a logical error. Allow only the owner of the `tokenIds` to execute a `refund` on them.