Metalabel Security Review

Pashov Audit Group

Conducted by: pashov

March 7th, 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 **Metalabel** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Metalabel

Copied from the first security review

Metalabel is a release club protocol. Groups of people with similar interests can gather and drop work together as collaborators. The protocol allows the creators to split the economic rewards that their "metalabel" has received amongst them. It has multiple technical abstractions like:

- Resources
    - Collection - ERC721 contract that mints tokens (records)
    - Split - a payment logic contract that has different percentage allocations for different contributors
    - Waterfall - a payment logic contract that enforces paying one party a certain amount before paying another party
- Accounts - for a user to unlock the functionalities of a protocol he needs to register an account (account creation is gated at first, later it becomes permissionless)
- Node - an ownable abstraction that groups Records and Resources and allows the owner account or a controller address to manage them
- Engine - a contract for dropping a new Collection, it manages mints, royalties, rendering (`tokenURI`) of the ERC721

The protocol is well-tested, as it has 100% code coverage (line, branch, function).

More docs

Continued here

The new version of the Metalabel protocol brings features that will improve the UX of the protocol, as well as `Memberships` functionality for minting NFTs to your squad. Some of the new abstractions are:

- Controller - a helper contract for setting up a new Metalabel and publishing a new release
- EngineV2 - comes with a built-in price decay mechanism, some metadata magic, optimizations and a fee for the owner
- Memberships - NFTs that show you are a part of a Metalabel

The newly added functionality is well-tested, as it also has 100% code coverage (line, branch, function).

More docs

Threat Model

Copied from the first security review

System Actors

- Account - can create a new Node
- Node owner - can manage nodes (configure collections, their mints, their royalties and price) and add controllers
- Controller - can manage nodes but can't add controllers
- Mint Authority - can mint permissioned sequences

External functions:

- `AccountRegistry` all methods - callable by anyone to register an account (unless `owner` is set)
- `NodeRegistry::createNode` - callable by anyone that creates an account
- `DropEngine::mint` - callable by anyone (unless `mintAuthorities` mapping is set for the sequence)

Q: What in the protocol has value in the market?

A: ERC721 token mints can be paid with ETH, so ETH value and also the ERC721 tokens themselves.

Q: What is the worst thing that can happen to the protocol?

1. Node ownership stolen
2. An attacker sets himself as mint payments' recipient
3. Exploiting the `mint` functionality so it allows free or unlimited mints

Interesting/unexpected design choices:

The owner of a group node can set multiple controllers for all other nodes in the group.

The controller of a group node can manage all other nodes in the group.

Controllers can be smart contracts, accounts can be smart contracts as well.

Controller of a node has the same rights as it's owner (apart from adding more controllers).

Continued here

Q: What is the worst thing that can happen to the protocol (which attack surface comes from this iteration)

1. Broken access control semantics in `ControllerV1` so catalog is ruined
2. Mint pricing errors in `DropEngineV2` based on price decay
3. Bot sniping whole sequence drops
4. Minter/user stealing mint fees for owner

Interesting/unexpected design choices:

The same instance of `ControllerV1` will be set as a controller to multiple nodes, so it will be shared.

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* - fb04291dfdf7114bbec12ef5ec30b4135eac4878**

Scope

The following smart contracts were in scope of the audit:

- `Memberships`
- `MembershipsFactory`
- `ControllerV1`
- `DropEngineV2`
- `RevenueModuleFactory`

7. Executive Summary

Over the course of the security review, pashov engaged with Metalabel to review Metalabel. In this period of time a total of **12** issues were uncovered.

Protocol Summary

Protocol Name	Metalabel
Date	March 7th, 2023

Findings Count

Severity	**Amount**
Medium	3
Low	4
QA	5
Total Findings	**12**

Summary of Findings

ID	Title	Severity	Status
[M-01]	The protection check for maxRecordsPerTransaction can be gamed	Medium	Resolved
[M-02]	Insufficient input validation opens up multiple attack vectors	Medium	Resolved
[M-03]	Owner can front-run sequence configurations by setting fee to 100%	Medium	Resolved
[L-01]	User can mint, burn and then re-mint his Memberships NFT	Low	Resolved
[L-02]	Anyone can mint your membership for you	Low	Resolved
[L-03]	No upper limit validation on user-supplied values	Low	Resolved
[L-04]	Merkle tree leaf generation is single-hashed and might lead to a second preimage attack if code changes	Low	Resolved
[QA-01]	Duplicated custom error	QA	Resolved
[QA-02]	NatSpecs are incomplete	QA	Resolved
[QA-03]	Redundant code	QA	Resolved
[QA-04]	Typos and grammatical errors in the comments	QA	Resolved
[QA-05]	Missing override keyword	QA	Resolved

8. Findings

8.1. Medium Findings

[M-01] The protection check for `maxRecordsPerTransaction` can be gamed

Severity

Impact: Medium, because a protocol invariant can be broken and the code gives a false sense of security

Likelihood: Medium, because the attack is easy to do and we have seen such attacks in the past

Description

Let's look at the following example scenario:

1. Collection creates a drop where `maxRecordsPerTransaction` is 1 and total supply is 10
2. It is expected that many people will be able to mint
3. A malicious actor writes a script that loads different wallets with enough value and bundles transactions for 10 mints
4. Only the malicious actor minted, no one else did

Even though there was some kind of a protection against bots/snipers the result was still that only 1 account got to minting.

Recommendations

Document that the `maxRecordsPerTransaction` check does not protect the protocol from sniping attacks. To protect from them you can decide to use an off-chain process for pre-registrations of addresses that will be put into a Merkle tree and then validated on `mint`.

[M-02] Insufficient input validation opens up multiple attack vectors

Severity

Impact: High, as it can overflow a balance and re-mint burned NFTs

Likelihood: Low, as it requires a malicious/compromised owner account or an owner input error

Description

The `adminTransferFrom` method does not validate that the `from` argument shouldn't have a value of `address(0)`. Now if `from == address(0)` multiple attack vectors open:

1. Burned NFTs can be "re-minted", and also that happens without changing `totalSupply`
2. If `from` balance is zero, this will underflow `_balanceOf[from]`
3. The 0 ID token (which shouldn't exist) can be "minted" with this method

Recommendations

Add a check and assert that the `from` argument is not `address(0)`.

[M-03] Owner can front-run sequence configurations by setting fee to 100%

Severity

Impact: High, as if it goes unnoticed it can rug the `revenueRecipient` address

Likelihood: Low, as it requires a malicious/compromised owner account

Description

The `setPrimarySaleFeeBps` is callable at any time by the contract owner address and will update the fee variable immediately. Now if a user is trying to

call `configureSequence`, the owner can front-run the user call, update the fee to 100% and since there is this code in `configureSequence`

```
dropData.primarySaleFeeBps = primarySaleFeeBps;
```

Now the whole mint payment for this sequence drop will go to the contract owner. He can also execute this attack and front-run each `configureSequence` call to get all mints' ETH value.

Recommendations

Since the user provides `dropData.primarySaleFeeBps`, check that he expected the same fee as the one that is currently set in `DropEngineV2` and if the current one is bigger revert the transaction. Also it is generally recommended to not allow fee to go up to 100% - lower the upper limit to a sensible number.

8.2. Low Findings

[L-01] User can mint, burn and then re-mint his `Memberships` NFT

The `mintMemberships` method allows multiple reuses of the same Merkle tree leaf, which means a user can mint, then burn, then mint again. This way he can spam events and also increase `totalMinted` and `totalSupply`. Add a check that forbids reuse of the same leaf in the Merkle tree.

[L-02] Anyone can mint your membership for you

The `mintMembership` method uses a Merkle tree that has the `mints[i].to` value in the leaf, instead of `msg.sender` - this means anyone can mint your membership for you. This means any user can influence the ID of the NFT which might not be desired. Prefer using `msg.sender` instead of a user-supplied value in the leaf generation.

[L-03] No upper limit validation on user-supplied values

The `decayStopTimestamp` and `priceDecayPerDay` properties of `DropData` in `DropEngineV2` do not have an upper limit validation. If too big values are set (due to an error for example) this can DoS the minting process. Add sensible upper limits for both values.

[L-04] Merkle tree leaf generation is single-hashed and might lead to a second preimage attack if code changes

Merkle trees whose leafs are just single-hashed are vulnerable to second preimage attack. The correct way is to double-hash them as OpenZeppelin suggests. The problem exists in both `ControllerV1` and in `Memberships` but in the latter the transaction would revert because of the max 1 balance check and in the former it will just setup a new Metalabel, but it can mess with the `subdomains` mapping.

8.3. QA Findings

[QA-01] Duplicated custom error

The `NotAuthorized` custom error is duplicated throughout the codebase. Declare it once and reuse it, same for duplicated interface which should be extracted in separate files.

[QA-02] NatSpecs are incomplete

@param and @return fields are missing throughout the codebase. NatSpec documentation is essential for better understanding of the code by developers and auditors and is strongly recommended. Please refer to the NatSpec format and follow the guidelines outlined there. Also the NatSpec of `createMemberships` is incorrect, same for `RevenueModuleFactory`.

[QA-03] Redundant code

The following imports are unused and can be removed `DropEngineV2` and `DropData` in `ControllerV1`, `MerkleProofLib` in `Memberships`. The `onlyAuthorized` modifier in `_mintAndBurn` can be removed because the functions that are calling this method already have it. Also `baseTokenURIs` and `mintAuthorities` are unused storage variables in `DropEngineV2` and should be removed.

[QA-04] Typos and grammatical errors in the comments

`Information provided with publishing a new release` -> Information provided when publishing a new release

`Lanch` -> `Launch`

`offchain` -> `off-chain`

`inheritted` -> `inherited`

`contorller` -> `controller`

`Tranfer` -> `Transfer`

`addidtional` -> `additional`

`iniitalization` -> `initialization`

`additonal` -> `additional`

`A an` -> `An`

`Admin can use a merkle root to set a large list of memberships that can /// minted by anyone with a valid proof to socialize gas` -> `that can be minted`

`Token URI computation defaults to baseURI + tokenID, but can be modified /// by a future external metadata resolver contract that implements IEngine` -> `ICustomMetadataResolver` instead of `IEngine`

`having to a separate storage` -> `having to do a separate storage`

`programatically` -> `programmatically`

`psuedo` -> `pseudo`

[QA-05] Missing `override` keyword

`DropEngineV2` inherits `configureSequence` method from `IEngine` but is missing the `override` keyword.