Metalabel Security Review

Pashov Audit Group

Conducted by: pashov

March 7th, 2023

Contents

1. About pashov 3
2. Disclaimer 3
3. Introduction 3
4. About Metalabel 4
5. Risk Classification 6
   - 5.1. Impact 6
   - 5.2. Likelihood 6
   - 5.3. Action required for severity levels 7
6. Security Assessment Summary 7
7. Executive Summary 8
8. Findings 10
   - 8.1. Medium Findings 10
     - [M-01] Insufficient input data validation for configureSequence 10
     - [M-02] Using the transfer function of address payable is discouraged 11
     - [M-03] Role transfer actions done in a single-step manner are dangerous 11
     - [M-04] Records minted to an address that is a smart contract that can't handle ERC721 tokens will be stuck forever 12
   - 8.2. Low Findings 14
     - [L-01] resolveId will return the 0 ID even if an address does not have an account 14
     - [L-02] Inconsistent tokenData.owner validation 14
   - 8.3. QA Findings 15
     - [QA-01] Use a more abstract name for function parameter 15
     - [QA-02] NodeData.nodeType is used as an existence check and nothing else 15
     - [QA-03] NatSpec docs are incomplete 15
     - [QA-04] Unused import 15
     - [QA-05] Copy-pasted comments in WaterfallFactory 15

[QA-06] Missing override keyword 16
[QA-07] Use a safe pragma statement 16
[QA-08] Typos and grammatical errors 16

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

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

Threat Model

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).

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* - effe2e89996d6809c5ec6c6da10c663246f91fc1**

Scope

The following smart contracts were in scope of the audit:

- `AccountRegistry`
- `Collection`
- `CollectionFactory`
- `DropEngine`
- `NodeRegistry`
- `Resource`
- `ResourceFactory`
- `SplitFactory`
- `WaterfallFactory`
- `interfaces/**`

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 **14** issues were uncovered.

Protocol Summary

Protocol Name	Metalabel
Date	March 7th, 2023

Findings Count

Severity	Amount
Medium	4
Low	2
QA	8
Total Findings	**14**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Insufficient input data validation for configureSequence	Medium	Resolved
[M-02]	Using the transfer function of address payable is discouraged	Medium	Resolved
[M-03]	Role transfer actions done in a single-step manner are dangerous	Medium	Resolved
[M-04]	Records minted to an address that is a smart contract that can't handle ERC721 tokens will be stuck forever	Medium	Resolved
[L-01]	resolveId will return the 0 ID even if an address does not have an account	Low	Resolved
[L-02]	Inconsistent tokenData.owner validation	Low	Resolved
[QA-01]	Use a more abstract name for function parameter	QA	Resolved
[QA-02]	NodeData.nodeType is used as an existence check and nothing else	QA	Resolved
[QA-03]	NatSpec docs are incomplete	QA	Resolved
[QA-04]	Unused import	QA	Resolved
[QA-05]	Copy-pasted comments in WaterfallFactory	QA	Resolved
[QA-06]	Missing override keyword	QA	Resolved
[QA-07]	Use a safe pragma statement	QA	Resolved
[QA-08]	Typos and grammatical errors	QA	Resolved

8. Findings

8.1. Medium Findings

[M-01] Insufficient input data validation for `configureSequence`

Severity

Impact: High, as some of those can result in a DoS or too big of a royalty payment

Likelihood: Low, as it requires a configuration error or a malicious actor

Description

An authorized address for a node can call `Collection::configureSequence` where most of the input is not validated properly. The `_sequence` parameter of the method is of type `SequenceData` which fields are not validated. Missing checks are the following:

1. `sealedBeforeTimestamp` is bigger than `block.timestamp`
2. `sealedAfterTimestamp` is always bigger than `sealedBeforeTimestamp`
3. The difference between `sealedBeforeTimestamp` and `sealedAfterTimestamp` is at least `2 days` for example
4. The difference between `sealedBeforeTimestamp` and `sealedAfterTimestamp` is not more than `500 days` for example
5. The difference between `sealedBeforeTimestamp` and `block.timestamp` is not more than `10 days` for example

Also in `DropEngine::configureSequence` the `royaltyBps` is not validated that it is not more than 100% (a value of 10000). I suggest you add a lower royaltyBps upper bound.

Recommendations

Add sensible constraints and validations for all user input mentioned above.

[M-02] Using the `transfer` function of `address payable` is discouraged

Severity

Impact: Medium, as sequence won't be usable as mints will revert

Likelihood: Medium, as it happens any time the recipient is a smart contract or a multisig wallet that has a receive function taking up more than 2300 gas

Description

The `mint` function in `DropEngine` uses the `transfer` method of `address payable` to transfer native asset funds to an address. This address is set by a node owner and is possible to be a smart contract that has a `receive` or `fallback` function that takes up more than the 2300 gas which is the limit of `transfer`. Examples are some smart contract wallets or multi-sig wallets, so usage of `transfer` is discouraged.

Recommendations

Use a `call` with value instead of `transfer`. There is no risk from reentrancy in the `mint` method as it has a check for the caller to be an EOA. When this is done you can remove the `payable` keyword from the `revenueRecipient` variable.

[M-03] Role transfer actions done in a single-step manner are dangerous

Severity

Impact: High, as important protocol functionality would become unusable

Likelihood: Low, as it requires an admin/owner error

Description

This is a common problem where transferring a role or admin rights to a different address can go wrong if this address is wrong and not actually controlled by any user. This is taken into consideration in `NodeRegistry` where the node ownership transfer is a two-step operation. Not the same approach is used in `AccountRegistry` though, where the contract inherits from `Owned` which has a single-step ownership transfer pattern and also the `transferAccount` logic in it is also using a single-step pattern.

Recommendations

Use a two-step ownership/rights transfer pattern in both the `AccountRegistry` ownership and in the `transferAccount` method, you can reuse the approach you used in `NodeRegistry`.

[M-04] Records minted to an address that is a smart contract that can't handle ERC721 tokens will be stuck forever

Severity

Impact: High, as records will be stuck forever

Likelihood: Low, as it requires the engine to allow smart contracts as minters and that contracts should not support handling of ERC721 tokens

Description

Both `mintRecord` methods in `Collection` use the `_mint` method of `ERC721` which is missing a check if the recipient is a smart contract that can actually handle ERC721 tokens. If the case is that the recipient can't handle ERC721 tokens then they will be stuck forever. For this particular problem the `safe` methods were added to the ERC721 standard and `Solmate` has added the `_safeMint` method to check handle this problem in a minting context. This is actually not a problem in `DropEngine` because it allows only EOAs to mint, but since users can freely implement Engines then this is a valid problem.

Recommendations

Prefer using `_safeMint` over `_mint` for ERC721 tokens, but do this very carefully, because this opens up a reentrancy attack vector. It's best to add a `nonReentrant` modifier in the method that is calling `_safeMint` because of this.

8.2. Low Findings

[L-01] `resolveId` will return the 0 ID even if an address does not have an account

This is error prone and requires all clients of the `resolveId` function to do checks for `!= 0`, as we can see in `NodeRegistry` where this method is called. A better and safer approach is to just revert in the `resolveId` method when the `subject` does not have an account, then if the client of the method wants to catch the error he can do this.

[L-02] Inconsistent `tokenData.owner` validation

The `mintRecord` functionality in `Collection` allows the caller to mint tokens to any address since the `to` argument is not validated in any way. The problem is that the `getTokenData` inherited function has the following check:

```
require(data.owner != address(0), "NOT_MINTED");
```

which is not actually true, because a token could be minted to the zero address, even due to a mistake. I suggest to add zero-address checks for the `to` argument in both `mintRecord` functions in `Collection` for consistency.

8.3. QA Findings

[QA-01] Use a more abstract name for function parameter

The `broadcast` and `broadcastAndStore` methods in `ResourceFactory` have a `waterfall` parameter but some other type of resource might be used - for example a Collection or a Split. Rename parameter from `waterfall` to `resource` in both methods.

[QA-02] `NodeData.nodeType` is used as an existence check and nothing else

The only thing `nodeType` is used for in the system is to check if a node exists. This can be done with a simple boolean or if you actually need a "type" then use an enum where the first value is `NON_EXISTENT` for example.

[QA-03] NatSpec docs are incomplete

In almost all methods the NatSpec documentation is incomplete as it is missing parameters and return variables in it. 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.

[QA-04] Unused import

The `IERC721` import in `Collection` is not used. Remove it from the code.

[QA-05] Copy-pasted comments in `WaterfallFactory`

Almost all methods reference a `split` instead of a `waterfall` in the contract, I expect they were copy-pasted from `SplitFactory`. Replace `split` with `waterfall` in all of the comments in `WaterfallFactory`.

[QA-06] Missing `override` keyword

Most methods in contracts that are implemented and inherited from an interface are missing the `override` keyword. Go through each method and apply the keyword where it is right to do so. Some examples are the `mintRecord` and `royaltyInfo` methods.

[QA-07] Use a safe pragma statement

Always use stable pragma statement to lock the compiler version and to have deterministic compilation to bytecode. Keep in mind `0.8.13` has a bug when using assembly, even though the codebase does not do that.

[QA-08] Typos and grammatical errors

A number of typos in the codebase that need to be fixed:

`AccountTransfered` -> `AccountTransferred`

`has be` -> `has been`

`Inheritting` -> `Inheriting`

`Otherise` -> `Otherwise`

`managaeable` -> `manageable`

`so long as their is` -> `as long as there is`

`entites` -> `entities`

`Otherise` -> `Otherwise`

`additonal` -> `additional`

`seqeuence` -> `sequence`

`funcionality` -> `functionality`

`reliquish` -> `relinquish`