IPNFT Security Review

Pashov Audit Group

Conducted by: pashov

June 9th, 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 **IPNFT** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About IPNFT

IPNFTs capture intellectual properties on the blockchain by using the ERC721 NFT standard. It implements a fractionalization mechanism which splits an IPNFT into a supply of ERC20 tokens, that can be issued or capped on-demand by the original owner of the IPNFT. The protocol also implements different types of crowd sales - a normal crowd sale, a crowd sale with vesting and a crowd sale with vesting + staking. They are used for fundraising governance shares (the fractionalized tokens) for a fixed price.

Crowd sales follow this order:

1. Configuring & starting a crowd sale, depositing the auction tokens
2. Accepting bids in bidding tokens until sale closes
3. Settling a sale - accounting and transferring the funding to the sale beneficiary
4. Claiming auction tokens for bidders

Observations

The `Fractionalizer` smart contract is non-custodial, meaning an address can call `fractionalizeIpnft` for a given IPNFT, then the address can transfer/sell the IPNFT and the new owner can also call `fractionalizeIpnft` on it.

The `IPNFT` contract is upgradeable, pausable and the owner can change the `mintAuthorizer` address on demand. This brings some centralization risk to the protocol.

When ot comes to the crowd sales logic, all types of ERC20's are expected except auction & staking tokens that don't have 18 decimals. Only bidding tokens can have != 18 decimals.

The crowd sales architecture is that `StakedVestedCrowdSale` inherits from `VestedCrowdSale` which inherits from `CrowdSale`. While each contract can be separately deployed, the team will only be personally deploying and supporting a `StakedVestedCrowdSale` contract.

A crowd sale can be oversold, bids are accepted until a sale closes, not until the funding goal has been reached.

Claiming auction tokens before the `closingTime + cliff` timestamp for a `VestedCrowdSale` or `StakedVestedCrowdSale` will result in gas overhead and tokens being locked in a `TimelockedToken` smart contract.

Currently it is possible that the `auctionToken` in a crowd sale is a `FractionalizedToken` that doesn't have a capped supply which means the original IPNFT owner can issue as many new fractions as he wants and dilute existing holders.

Threat Model

Privileged Roles & Actors

- IPNFT original owner - an entity that can issue token fractions and dilute existing token holders. It can also cap the issuance of new fractions.
- FractionalizedToken owner - always the `Fractionalizer` contract, can issue new fractions or cap the supply
- FractionalizedToken issuer - the original owner of the fractionalized IPNFT, can issue new fractions or cap the supply, same user who calls `fractionalizeIpnft`
- Fractionalizer owner - can set the fee receiver address and the fee percentage, also can authorize upgrades to the `Fractionalizer` contract
- IPNFT owner - can pause-unpause the `IPNFT` contract, set its `mintAuthorizer` address, withdraw its fees or authorize an upgrade on it
- Mint authorizer - authorizes reservations & mints for IPNFTs
- CrowdSale starter - can start and configure a crowd sale
- CrowdSale bidder - can deposit `biddingToken`s to a crowd sale with the goal of attaining `auctionToken`s

Security Interview

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

A: The IP rights in the IPNFTs as well as the `auctionToken`s, `biddingToken`s and `stakingToken`s in crowd sales.

Q: In what case can the protocol/users lose money?

A: If they can't claim tokens from a `CrowdSale` or if their `IPNFT` has incorrect IPs.

Q: What are some ways that an attacker achieves his goals?

A: Claim more `auctionToken`s than he should had from a `CrowdSale` or make other users' claims revert.

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 **- c99cbf42f10c28ca5912c8f44f1ebf48660826a7**

fixes review commit hash **- a56b9302c74d73949b586f50cc1ed17a2f2bff49**

Scope

The following smart contracts were in scope of the audit:

- `crowdsale/CrowdSale`
- `crowdsale/StakedVestedCrowdSale`
- `crowdsale/VestedCrowdSale`
- `Fractionalizer`
- `FractionalizedToken`
- `TimelockedToken`
- `IPNFT`

7. Executive Summary

Over the course of the security review, pashov engaged with IPNFT to review IPNFT. In this period of time a total of **9** issues were uncovered.

Protocol Summary

Protocol Name	IPNFT
Date	June 9th, 2023

Findings Count

Severity	**Amount**
High	1
Medium	3
Low	5
Total Findings	**9**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Sale creator can possibly steal bidders' claimable tokens	High	Resolved
[M-01]	Auction tokens with approval race protection or not returning a bool on approve are incompatible with VestedCrowdSale	Medium	Resolved
[M-02]	Insufficient input validation in CrowdSale configuration	Medium	Resolved
[M-03]	Tokens with a fee-on-transfer mechanism will break the protocol	Medium	Resolved
[L-01]	Round down instead of up for token claims	Low	Resolved
[L-02]	Use pull over push pattern in CrowdSale::settle	Low	Resolved
[L-03]	Usage of address payable's send method is discouraged	Low	Resolved
[L-04]	Percentage setter should be bounded	Low	Resolved
[L-05]	Correct state is not enforced in CrowdSale::claim	Low	Resolved

8. Findings

8.1. High Findings

[H-01] Sale creator can possibly steal bidders' claimable tokens

Severity

Impact: High, as it results in a loss of funds for bidders

Likelihood: Medium, as it requires to claim before cliff expires

Description

In `StakedVestedCrowdSale` the sale creator can give the address to his own `TokenVesting` & `TimelockedToken` contracts. Same in `VestedCrowdSale` but only for `TimelockedToken`. Now if the sale creator is malicious he can give the addresses of his own deployed contracts that inherit from either `TokenVesting` or `TimelockedToken` but add functionality to pull the funds out on demand, while they are still locked in them. This is even worse when it comes to the token locking logic in `VestedCrowdSale`, where on sale settlement the `TimelockedToken` contract is approved to spend all the `auctionToken` that should be claimed, meaning if it has the functionality it can just pull the funds and transfer them out of the contract on demand. This will result in inability for bidders to claim their tokens and 100% loss of their value.

Recommendations

Enforce both `TokenVesting` & `TimelockedToken` contracts to be only internally deployed from a predefined bytecode/implementation and do not accept user-supplied contracts.

8.2. Medium Findings

[M-01] Auction tokens with approval race protection or not returning a `bool` on `approve` are incompatible with `VestedCrowdSale`

Severity

Impact: High, as sale won't be possible to be settled

Likelihood: Low, as such tokens are rare, but they do exist

Description

Some tokens, for example USDT (not a good example because auction tokens have to have 18 decimals) and KNC (18 decimals) have approval race protection mechanism and require the allowance to be either 0 or `uint256.max` when it is updated. The problem is that in `VestedCrowdSale` the following code is present:

```
uint256 currentAllowance = sale.auctionToken.allowance(address(this), address
  (salesVesting[saleId].vestingContract));

//the poor man's `increaseAllowance`
sale.auctionToken.approve(address
  (salesVesting[saleId].vestingContract), currentAllowance + sale.salesAmount);
```

This will not work with tokens that have such a mechanism and will revert when `currentAllowance` is non-zero.

Another issue is that there are tokens that do not follow the ERC20 standard (like `USDT` again) that do not return a `bool` on `approve` call. Those tokens are incompatible with the protocol because Solidity will check the return data size, which will be zero and will lead to a revert.

Recommendations

Do the approvals only in `VestedCrowdSale::_claimAuctionTokens` just before tokens are about to be locked, or just approve to zero first. Make sure to use `forceApprove` from `SafeERC20`.

[M-02] Insufficient input validation in `CrowdSale` configuration

Severity

Impact: High, as it can lock up valuable tokens almost permanently

Likelihood: Low, as it requires a fat-finger or a big configuration error

Description

There are two flaws in the configuration validation of a new `CrowdSale`. It is currently possible to create a never ending `Sale` as the `closingTime` field does not have a max value check. It is also possible that a never ending lock or a 0 duration lock is used in `VestedCrowdSale` as the `cliff` argument of `startSale` is not validated as in `StakedVestedCrowdSale::startSale`. Both can result in almost permanently locked tokens which is a value loss for users.

Recommendations

Add proper min & max value bounds for both `closingTime` and `cliff` parameters.

[M-03] Tokens with a fee-on-transfer mechanism will break the protocol

Severity

Impact: High, as some users will lose value

Likelihood: Low, as such tokens are not common

Description

The ERC20 logic in all crowd sale contracts as well as in `TimelockedToken` is incompatible with tokens that have a fee-on-transfer mechanism. Such tokens for example is `PAXG`, while `USDT` has a built-in fee-on-transfer mechanism that is currently switched off. One example of this `CrowdSale::startSale` where the following code:

```
sale.auctionToken.safeTransferFrom(msg.sender, address(this), sale.salesAmount);
```

Will work incorrectly if the token has a fee-on-transfer mechanism - the contract will cache `sale.salesAmount` as it's expected balance, but it will actually have `sale.salesAmount - fee` balance. This will result in a revert in the last person to transfer `auctionTokens` out of the contract. Same thing applies for other `transferFrom` calls that transfer tokens into the protocol, for example in `TimelockedToken::lock`.

Recommendations

You should cache the balance before a `transferFrom` to the contract and then check it after the transfer and use the difference between them as the newly added balance. This also requires a `nonReentrant` modifier, as otherwise ERC777 tokens can manipulate this. Another fix is to just document and announce you do not support tokens that can have a fee-on-transfer mechanism.

8.3. Low Findings

[L-01] Round down instead of up for token claims

Currently all the `wad` math in `CrowdSale`s contracts is rounding up. This is not good for token claims as if some user claims 1 wei of token more than he should had it is possible that for the final user who claims to get to a revert. Make sure to round down instead of up for token claims.

[L-02] Use pull over push pattern in `CrowdSale::settle`

The `settle` method in `CrowdSale` is directly transferring funds to the `sale.beneficiary`, in `SaleState.FAILED` case `auctionToken`s and in `SaleState.SETTLED` `biddingToken`s. If either of those tokens has a blacklisting mechanism and the beneficiary has been blacklisted, this will result in all `auctionToken`s and `biddingToken`s in the sale being stuck in it, as there will be no way to `settle` the sale. You can use the pull over push pattern to resolve this issue, for example by not force-transferring the funds to the `sale.beneficiary`, but adding a method through which he should claim them.

[L-03] Usage of `address payable`'s `send` method is discouraged

The `send` method of `address payable` is deprecated as if the callee has a `receive` or `fallback` function that takes up more than 2300 gas then the method will revert. Since there are many smart wallets or multi-sig wallets that take more than 2300 gas on native asset receival, it is recommended to use `call` with value instead

```
-require(payable(_msgSender()).send(address(this).balance), "transfer failed");
+msgSender().call{value: address(this).balance}("");
```

[L-04] Percentage setter should be bounded

The `Fractionalizer::setReceiverPercentage` method sets a percentage value that is unbounded, meaning it can possibly be more than 100%. While this functionality is still not used it seems dangerous. I suggest you remove it until it is used (together with `setFeeReceiver`), or add a `MAX_VALUE` constant to cap the percentage value possible.

[L-05] Correct state is not enforced in `CrowdSale::claim`

The `claim` method in `CrowdSale` will revert if a sale is still in `RUNNING` state and will go through the `claimFailed` route if it is in `FAILED` state. The problem is that it doesn't enforce a revert if the state after the first checks is not `SETTLED`. This currently is not problematic but can lead to subtle bugs in the future, like claiming for a sale that is neither `FAILED` or `SETTLED`.