Bear Cave Security Review

Pashov Audit Group

Conducted by: pashov

May 17th, 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 **Bear Cave** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Bear Cave

Bear Cave is an NFT protocol that hosts a number of games on-chain. Each different bear (NFT token from a bear collection) has a set of "gates" through which it can get into the protocol.

These are the core contracts in the protocol:

- `Gatekeeper`, which manages the various gates
- `GameRegistry`, which manages game stages and permissions
- `HoneyJar`, which is a simple ERC721 compliant NFT contract
- `HoneyBox`, which mints `HoneyJar` NFTs and manages bundles (groups of ERC721/ERC1155 collections)
- `HoneyJarPortal`, which allows for cross-chain interactions with the `HoneyJar` NFTs

More docs

Observations

There are five mechanisms for players to mint `HoneyJar` tokens through `HoneyBox`:

1. Claim them for free, if included in free mint whitelist
2. Early paid mint with `paymentToken`, if included in early paid mint whitelist
3. Early paid mint with ETH, if included in early paid mint whitelist
4. Public paid mint with `paymentToken`
5. Public paid mint with ETH

After the `mintConfig.maxHoneyJar` amount is minted, the `_findHoneyJar` method is called, which does a VRF request and chooses the "special Honey Jar", whose owner can claim the `sleepoor` NFTs in a specific bundle.

Threat Model

Privileged Roles & Actors

Privileged Roles

- Game admin - can control and configure most of the protocol. Granted to the Bear Cave team's multi-sig
- Game instance - updates `Gatekeeper`'s internal claimed `HoneyJar` tokens accounting and starts gates for a token. Granted to the `HoneyBox` contract
- Minter - can mint `HoneyJar` tokens. Granted to both the `HoneyBox` and `HoneyJarPortal` contracts
- Burner - can burn any `HoneyJar` token, no matter who owns it. Currently not granted to anyone, it is expected that it will be granted to the `HoneyBox` contract

The Game admin can grant the `GAME_INSTANCE` and `MINTER` roles to any address.

Actors

- Beekeeper - receives a share of the `HoneyJar` NFT mint fees
- Jani - receives a share of the `HoneyJar` NFT mint fees
- Player - can claim free `HoneyJar` tokens based on eligibility or mint `HoneyJar` tokens with ERC20 tokens or ETH
- VRF - supplies randomness so that the "special Honey Jar" NFT is chosen.

Security Interview

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

A: The NFTs that are deposited into the `HoneyBox` contract. Also the "special" `HoneyJar` NFT is worth at least the value of all the NFTs stored in the `HoneyBox` contract. The `HoneyJar` NFTs are valuable as well, as they cost ERC20 tokens or ETH to mint.

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

A: If the game never ends or if an attacker games the minting mechanics or the "special" `HoneyJar` VRF logic.

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

A: Making some of the method's logic revert by underflow/overflow or force an external call to revert. Also exploit the VRF integration, or the minting mechanics.

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* - 6c098a53649c2cf08afc806be37f9d50835a5252**

Scope

The following smart contracts were in scope of the audit:

- `Constants`
- `GameRegistry`
- `GameRegistryConsumer`
- `GateKeeper`
- `HoneyBox`
- `HoneyJar`
- `HoneyJarPortal`
- `IHoneyJar`

7. Executive Summary

Over the course of the security review, pashov engaged with Bear Cave to review Bear Cave. In this period of time a total of **18** issues were uncovered.

Protocol Summary

Protocol Name	Bear Cave
Date	May 17th, 2023

Findings Count

Severity	**Amount**
Critical	4
High	2
Medium	2
Low	4
QA	6
Total Findings	**18**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Anyone can steal all HoneyJar NFTs in HoneyJarPortal and exploit its allowances	Critical	Resolved
[C-02]	Reentrancy allows any user allowed even one free HoneyJar mint to mint the max supply for himself for free	Critical	Resolved
[C-03]	Anyone can mint all NFTs through the public mint before it has even started	Critical	Resolved
[C-04]	Re-requesting randomness from VRF is a security anti-pattern	Critical	Resolved
[H-01]	Mint function mekHoneyJarWithETH will revert every time	High	Resolved
[H-02]	Admin account has a lot of power in the protocol and multiple ways to deny/steal users' rewards	High	Resolved
[M-01]	Possible overflow will break the logic in HoneyBox	Medium	Resolved
[M-02]	Multiple flaws in the gate reset logic in Gatekeeper	Medium	Resolved
[L-01]	The BURNER role and burn method are not usable	Low	Resolved
[L-02]	Discrepancy between implementation and docs	Low	Resolved
[L-03]	The Checks-Effects-Interactions pattern is not followed	Low	Resolved
[L-04]	Insufficient validation in multiple Gatekeeper Methods	Low	Resolved

[QA-01]	Incomplete NatSpecs	QA	Resolved
[QA-02]	Unused or redundant code can be removed	QA	Resolved
[QA-03]	Typos and grammatical errors in the code	QA	Resolved
[QA-04]	Use the complete name of types	QA	Resolved
[QA-05]	Missing override keyword	QA	Resolved
[QA-06]	Missing event emissions in state changing methods	QA	Resolved

8. Findings

8.1. Critical Findings

[C-01] Anyone can steal all `HoneyJar` NFTs in `HoneyJarPortal` and exploit its allowances

Severity

Impact: High, as it is a value loss for users

Likelihood: High, as it is a common vulnerability and requires no preconditions

Description

The `_debitFrom` function in `HoneyJarPortal` is exploitable, as it looks like this:

```
function _debitFrom
  (address _from, uint16, bytes memory, uint _tokenId) internal override {
    honeyJar.safeTransferFrom(_from, address
    //(this), _tokenId); // Performs the owner & approval checks
}
```

Since there is no check for the `_from` argument, anyone can call the function (through the `sendFrom` method in `ONFT721Core`) and pass the address of `HoneyJarPortal` as the `_from` argument and his address as the `_toAddress` argument in the `sendFrom` method and essentially steal every NFT that is owned by the `HoneyJarPortal`. It can also steal NFTs that `HoneyJarPortal` does not own, but is an approved spender of, since the `safeTransferFrom` method will complete successfully.

Recommendations

In `_debitFrom` check that the owner of the `_tokenId` NFT is the `msg.sender`.

[C-02] Reentrancy allows any user allowed even one free `HoneyJar` mint to mint the max supply for himself for free

Severity

Impact: High, as the user will steal all `HoneyJar` NFTs, paying nothing

Likelihood: High, as reentrancy is a very common attack vector and easily exploitable

Description

The `claim` method in `HoneyBox` (from its NatSpec) "Allows a player to claim free HoneyJar based on eligibility". Let's look at this part of its code:

```
_canMintHoneyJar
//(bundleId_, numClaim); // Validating here because numClaims can change

// If for some reason this fails, GG no honeyJar for you
_mintHoneyJarForBear(msg.sender, bundleId_, numClaim);

claimed[bundleId_] += numClaim;
// Can be combined with "claim" call above, but keeping separate to separate
// view + modification on gatekeeper
gatekeeper.addClaimed(bundleId_, gateId, numClaim, proof);
```

Where you update the `claimed` mapping and account for the claim in the `Gatekeeper` contract after you actually do the minting itself. The problem is that the `_mintHoneyJarForBear` method calls `honeyJar::batchMint`, that uses `safeMint`, which does an unsafe external call to the mint recipient. This call can reenter the `claim` method while the `claimed` accounting was still not done and actually claim all of the `HoneyJar` NFTs until `mintConfig.maxHoneyJar` is hit, which will most likely make him the winner of the game so he will get all of the NFTs in it as well, paying nothing.

What makes it worse as well is that even though the `claim` method has protection because it accepts a `gateId` argument, and the gates themselves have a `maxClaimable` property, this is also broken since the `gatekeeper::addClaimed` call is also done after the unsafe external call, so multiple invariants can be broken here.

Recommendations

Make sure the `claim` method is following the Checks-Effects-Interactions pattern or add a `nonReentrant` modifier to it.

[C-03] Anyone can mint all NFTs through the public mint before it has even started

Severity

Impact: High, as it breaks an important protocol invariant and the way the protocol should work overall

Likelihood: High, as it does not need any preconditions, can be executed easily at the deployment of `HoneyBox`

Description

Both the `mekHoneyJarWithERC20` and `mekHoneyJarWithETH` methods are ways for the players to mint `HoneyJar` NFTs, but they should work only when general mint is open, as shown in this check that is present in both methods:

```
if (slumberParties[bundleId_].publicMintTime > block.timestamp)
    revert GeneralMintNotOpen(bundleId_);
```

The problem is that anyone can call both methods anytime before the first bundle was added. If there were no bundles, this means that if a user supplied `bundleId_ == 0` to either method, all of the values in the `slumberParties[bundleId_]` mapping will have a default value, passing all of the checks in the methods and in the `_canMintHoneyJar` method. This essentially means anyone can front-run the games and mint the maximum available `HoneyJar` configured in the `mintConfig`.

Recommendations

In `_canMintHoneyJar`, revert if `slumberParties[bundleId_].publicMintTime == 0`, this means that this bundle is not initialized yet. This will cover this attack vector and any other `bundleId == 0` attack as well.

[C-04] Re-requesting randomness from VRF is a security anti-pattern

Severity

Impact: High, as the VRF service provider has control over who wins the game

Likelihood: High, as there is an incentive for a VRF provider to exploit this and it is not hard to do from his side

Description

The `forceHoneyJarSearch` method is used to "kick off another VRF request", as mentioned in its NatSpec. This goes against the security standards in using VRF, as stated in the docs:

```
Re-requesting randomness is easily detectable
on-chain and should be avoided for use cases
that want to be considered as using VRFv2 correctly.
```

Basically, the service provider can withhold a VRF fulfillment until a new request that is favorable for them comes.

Recommendations

Remove the `forceHoneyJarSearch` method as it is exploitable.

8.2. High Findings

[H-01] Mint function `mekHoneyJarWithETH` will revert every time

Severity

Impact: Medium, as there is an option to mint with ERC20 tokens too

Likelihood: High, as the function will just revert every time

Description

The `HoneyBox` contract exposes a way for users to mint `HoneyJar` NFTs with `ETH` in a public sale by the `mekHoneyJarWithETH` method. The problem is that the method uses `msg.value` to calculate the expected price, as the name suggest, that would have been paid with ETH, but the method is missing the `payable` keyword. Every call with `msg.value != 0` to the method will revert.

Recommendations

```diff
-    function mekHoneyJarWithETH
- (uint8 bundleId_, uint256 amount_) external returns (uint256) {
+    function mekHoneyJarWithETH
+ (uint8 bundleId_, uint256 amount_) external payable returns (uint256) {
```

[H-02] Admin account has a lot of power in the protocol and multiple ways to deny/steal users' rewards

Severity

Impact: High, as the admin can steal funds from users(players)

Likelihood: Medium, as it requires a malicious or a compromised admin, but the incentives are high

Description

There are multiple centralization flaws and attack vectors in the protocol:

- Game admin can make `openHotBox` revert by front-running it with a `forceHoneyJarSearch` call, essentially rugging game winner
- Game admin can call `setVRFConfig` and use his personally controlled VRF interface compliant contract and use a non-random number, essentially deciding the outcome of the game
- The `setMaxhoneyJar` method can be used so the game never finishes. Even though it has the `isEnabled` check, and there is a comment saying `should not be called while a game is in progress to prevent hostage holding.` this is not true, as the same address (`GAME_ADMIN` role) that can call `setMaxhoneyJar` can call `startGame` and `stopGame` which set the `enabled` flag, so the admin can set the max to a huge number, essentially putting the game into a state of DoS
- Since game admin controls when a game is started or stopped, it can also front-run users with a call to `setHoneyJarPrice_ERC20` and make them pay more if they put unlimited allowance, essentially stealing their tokens

Recommendations

Redesign all methods that can be used as rug pulls and possibly make the admin in the protocol a Timelock contract.

8.3. Medium Findings

[M-01] Possible overflow will break the logic in `HoneyBox`

Severity

Impact: High, as bundles storage variables will be overwritten

Likelihood: Low, as it is not expected to add more than 255 bundles

Description

In `HoneyBox::addBundle` we have the following code:

```
// Will fail if we have >255 bundles
uint8 bundleId = uint8(slumberPartyList.length);
```

The comment is wrong, as it assumes that the cast is safe and will revert if `slumberPartyList.length > 255` but this is not the case as it will just overflow. This will be a big problem as then already existing `bundleId` values will be overwritten in the `slumberParties` mapping, which will break the logic of the contract.

Recommendations

Use a `SafeCast` library or revert if `slumberPartyList.length > 255`.

[M-02] Multiple flaws in the gate reset logic in `Gatekeeper`

Severity

Impact: Medium, as no value will be lost but the contract state will be incorrect

Likelihood: Medium, as it is not expected to happen every time, but there are multiple attack paths here

Description

The `resetAllGates` method is iterating over unbounded arrays - both `tokenToGates[tokenId]` and `consumedProofsList[gateId]` arrays are unbounded. This might result in a state of DoS for the `resetAllGates` method, since it might take too much gas to iterate over the arrays (more than the block gas limit).

Another, bigger problem in the method, is that it does not do `delete` on `tokenToGates[tokenId]` - even though it sets `claimedCount` to 0, it does not set `claimed` to `false` for example, so methods that check this will still think that `claimed == true` (for example `validateProof` checks it).

Recommendations

Make sure to add an upper bound to both `tokenToGates[tokenId]` and `consumedProofsList[gateId]` arrays size, in the `addGate` and `addClaimed` methods respectively. Make sure to call `delete` on `tokenGates[i]` in the first `for` loop in `resetAllGates` and also emit an `GateReset` event for each reset gate in the `resetAllGates` method.

8.4. Low Findings

[L-01] The `BURNER` role and `burn` method are not usable

The `burn` method is not called anywhere, and there is no option for anyone to be granted the `BURNER` role currently. As discussed with the team, this means the code will be changing in the near future. This holds risk and I recommend either removing the `BURNER` role & `burn` functionality or getting a new audit with the new code included.

[L-02] Discrepancy between implementation and docs

The NatSpec docs of `registerGame` say that the method "enables" a game, but the `games` mapping is not actually set to `true` for that game (this is done in `startGame`). This is misleading and can lead to errors - either update the implementation or the NatSpec accordingly.

[L-03] The Checks-Effects-Interactions pattern is not followed

The methods `puffPuffPassOut` and `openHotBox` in `HoneyBox` are not following the CEI pattern. Even though they have no reentrancy vulnerability right now, it is recommended to follow the CEI pattern for security, as it is possible that the code changes with time.

[L-04] Insufficient validation in multiple `Gatekeeper` Methods

The `addClaimed` method is missing the `gate.enabled` and `gate.activeAt` checks that are present in `validateProof`, so they should be added. It should also revert if `consumedProofs` mapping is already set to `true` for complete correctness.

The `addGate` method should check if `tokenId` exists and also the `maxClaimable_` argument should have an upper bound.

The `startGatesForToken` should check if a gate is already enabled and revert if that is the case, as the current implementation allows for a "restart" of a gate for a `tokenId`.

8.5. QA Findings

[QA-01] Incomplete NatSpecs

Most `external` methods in the codebase have a NatSpec documentation but they almost always lack `@param` and `@return` explanation. Short example is `HoneyBoxx::addBundle` - neither the parameters or the return variable are listed in the NatSpec. Go through all methods and make sure they have a complete and proper documentation.

[QA-02] Unused or redundant code can be removed

The `GATEKEEPER` constant in `Constants` is not used anywhere and should be removed

The `initialize` method in `HoneyBox` is not needed, as it is not upgradeable and does not implement any proxy functionality. Move its logic to the contract's constructor. Also remove the `NotInitialized` and `AlreadyInitialized` errors, the `Initialized` event and the `initialized` storage variable.

The following imports in `Gatekeeper` are not used and should be removed:

- `import {ERC1155} from "solmate/tokens/ERC1155.sol";`
- `import {ERC20} from "solmate/tokens/ERC20.sol";`
- `import {ERC721} from "solmate/tokens/ERC721.sol";`

The `BEEKEEPER` & `JANI` roles and their setters and constants are not used and not needed, can be removed.

The `games` storage mapping in `Gatekeeper` is not used anywhere and should be removed.

The `// Switching to OZ for LZ compatibility` comment is not needed, as previous code is kept in the version control system anyway.

The `if (mintAmount == 0) revert ZeroMint();` check in `earlyMekHoneyJarWithERC20` can be removed, because the check is present in `_canMintHoneyJar` which is called by `earlyMekHoneyJarWithERC20`.

The `honeyJarToParty` mapping in `HoneyBox` is only written to, never read from on-chain. This wastes gas and should be removed.

Redundant comment - `// Save to mapping`. It does not add value and can be removed.

Remove the `if (proof[i].length == 0) continue; // Don't nomad yourself` check in `HoneyBox::claimAll` since you call `claim` after it, and in `claim` you have `if (proof.length == 0) revert Claim_InvalidProof();`

[QA-03] Typos and grammatical errors in the code

There are multiple typos and grammatical errors throughout the codebase

- `elegibility` -> `eligibility`
- `logic is affects` -> `logic is affected`
- `you a player wants to claim` -> `a player wants to claim`
- `mintor` -> `minter`
- `All game contracts should use extend` -> `All game contracts should extend`
- `all free cams` -> `all free claims`

[QA-04] Use the complete name of types

Replace `uint` with `uint256` everywhere in the codebase, it is seen in `HoneyJarPortal` for example. Subtle bugs can slip if you just use `uint`, for example if you are hashing the signature of a method and use `uint` instead of `uint256` for any parameter type.

[QA-05] Missing `override` keyword

The `HoneyJar` contract is implementing the `IHoneyJar` interface but is lacking the `override` keyword in multiple places. Add it to the `mintTokenId`, `batchMint` and `burn` methods as well as the `nextTokenId` storage variable, to make use the compiler checks from the keyword.

[QA-06] Missing event emissions in state changing methods

It's a best practice to emit events on every state changing method for off-chain monitoring. The `setBaseURI` and `setGenerated` methods in `HoneyJar` are missing event emissions, which should be added.