



# PopBlocks Trashbin.sol Review

Nov 9, 2022

# Table of Contents





<b>Disclaimer</b>	<b>3</b>
<b>Introduction</b>	<b>4</b>
Definitions	4
Summary	4
Review Details	4
Issues Overview	4
Review Checklist	5
<b>Review</b>	<b>6</b>
1. Smart Contract Description	6
2. Business Requirements Review	6
3. Standards and Best Practice Review	8
4. Exaggerated Owner Rights	9
5. Potential Exploits	10
6. Recommendations	11

# Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions.

# Introduction

## Definitions

-  - Statement was checked and passed the review.
-  - Major issue that needs to be addressed.
-  - Medium issue that requires attention of the developer(s).
-  - Recommendation

## Summary




The purpose of this review was to achieve the following:

1. Identify potential security issues with the TrashBin smart contract.
2. Formally check the logic behind the given TrashBin smart contract.

## Review Details

<b>Time of Review</b>	7 Nov 2022 - 9 Nov 2022
<b>Contract Name</b>	TrashBin.sol
<b>MD5 Contract Sum</b>	4b49444826e5def6e7220a10319ffa7d

## Issues Overview

Type	Count
 Major Issue	3
 Medium Issue	10
 Recommendation	3

## Review Checklist

- Business logic overview
- Exaggerated owner permissions
- Setters allow values that could break business logic
- Gas optimisation
  - Unnecessary reads from storage
  - External calls to itself
- Maths
  - Division before multiplication
- Spelling
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC and other standards violation


# Review


## 1. Smart Contract Description


The TrashBin smart contract aims to facilitate writing off losses on NFT investments by providing a guaranteed purchase of NFT tokens for a low price.


## 2. Business Requirements Review


### TrashBin Sell Logic


2.1  An ERC721 token can be sold using ***TrashBin.sell()*** for ***100 Mgwei***

2.2  An ERC1155 token can be sold using ***TrashBin.sell()*** for ***100 Mgwei***

2.3  Multiple ERC721 tokens can be sold using ***TrashBin.sell()*** for ***100 Mgwei \* amountOfTokens***

2.3  Multiple ERC1155 collections can be sold using ***TrashBin.sell()*** for ***100 Mgwei \* amountOfERC1155CollectionIds***

 The implicit ignorance of the the ERC1155 collection token amounts (e.g. ERC1155 collection id 1 with 500 value) can confuse users. Users could expect to receive ***500 \* 100 Mgwei*** but will only receive ***100 Mgwei*** . This could lead to complaints.

 A savvy user can split the amounts of an ERC1155 collection and thus achieve receiving ***100Mgwei \* splitCount***. A decision needs to be made on whether payouts should be exercised based on unique contract addresses or not. (see also point 2.6). Example:  
***TrashBin.sell([0x1, 0x1, ..., 0x1], [1, 1, ..., 1], [1, 1, ..., 1], [true, ...]);***

## TrashBin Transfer Logic

2.4 ✓ Transferring one ERC721 token to TrashBin will transfer the sender **100 Mgwei**

2.5 ✓ Transferring one ERC1155 collection with any amount greater than 1 will transfer sender **100 Mgwei**.

⚠ The implicit ignorance of the the ERC1155 collection token amounts (e.g. ERC1155 collection id 1 with 500 value) can confuse users. Users could expect to receive  $500 * 100$  Mgwei but will only receive 100 Mgwei. This could lead to complaints.

2.6 ! Transferring multiple ERC1155 collections with any amount greater than 1 will transfer the sender **100 Mgwei \* amountOfERC1155CollectionIds** which does not comply with the sell business logic.

### Method with issue: onERC1155BatchReceived

- [https://github.com/CookedCookee/FragmentsDAO/blob/master/contracts/ashes\\_collections/trashbin/TrashBin.sol#L587](https://github.com/CookedCookee/FragmentsDAO/blob/master/contracts/ashes_collections/trashbin/TrashBin.sol#L587)

Line issue: `(bool success, ) = (from).call{value: sellPrice}("");`

- [https://github.com/CookedCookee/FragmentsDAO/blob/master/contracts/ashes\\_collections/trashbin/TrashBin.sol#L610](https://github.com/CookedCookee/FragmentsDAO/blob/master/contracts/ashes_collections/trashbin/TrashBin.sol#L610)

### Suggested remedy:

- Multiply the sellPrice by the amount of ERC1155 collection ids (or unique ones, see 2.3).

## TrashBin Buy Logic

2.7 ✓ Buying one ERC721 token by paying **0.02 ETH**

2.8 ✓ Buying multiple ERC721 tokens by paying **0.02 ETH \* amountOfTokens**

2.9 ✓ Buying one ERC1155 collection token by paying **0.02 ETH**

2.10 ✓ Buying multiple ERC1155 collection tokens by paying **0.02 ETH \* amountOfTokens**

2.11 ✓ Buying multiple various ERC721 and ERC1155 tokens by paying **0.02 ETH \* amountOfTokens**

2.12 ✓ Applying discount prices for hashDAO token owners and hasDAO governance token owners.

### 3. Standards and Best Practice Review

3.1 ⚠ Owner and HashDAO protected methods are not emitting events and thus affect transparency.

**Example of methods with issue:**

- *withdrawETH*
- *removeNFT*
- *withdrawERC721*
- *withdrawERC1155*
- *withdrawERC20*
- *deleteIndex*
- *updateAllSettings*
- *\_updateBuyPrice*
- *\_updateSellPrice*
- *\_updateStandardHashDiscount*
- *\_updateDaoHashDiscount*
- *\_updateBuyableDelay*
- *\_updateMinEthBalance*
- *\_updateEthPercentage*

**Suggested Remedy:**

- Add corresponding events to the above mentioned methods.

3.2 ⚠ Not using the latest solidity compiler version, currently ***pragma solidity ^0.8.6***.

**Suggested Remedy:**

- Use latest solidity version ***v0.8.17***

3.3 ⚠ ***isNFTAvailableToBuy*** does not check if index is not out of bounds and reverts with panic code 50.

**Suggested Remedy:**

- Add a dedicated revert reason for index out bounds, such as to inform the user that an invalid index was used.



3.4 ⚠ Following methods allow looping through unlimited iterations over array elements.

While very low risk this could potentially expose the contract to DoS attacks to block other users from buying a valuable NFT until the buyableDelay is reached. Thus allowing the attacker to buy the NFT.

Reference:

<https://consensys.github.io/smart-contract-best-practices/attacks/denial-of-service/#gas-limit-dos-on-the-network-via-block-stuffing>

**Affected methods:**

- *buy*
- *sell*
- *onERC1155BatchReceived*

**Suggested Remedy:**

- Define a limit for the passed array sizes and/or set a gas limit for the loops (msg.gas)

3.5 ⚠ Multiple reads from storage that impact gas.

**Suggested Remedy:**

- *withdrawETH - address(this).balance* is called twice. Consider saving the balance into a variable and reusing it.

## 4. Exaggerated Owner Rights

4.1 ⚠ Arbitrary value can be set on the contract settings.

**Affected Methods:**

- *\_updateBuyPrice*
- *\_updateSellPrice*
- *\_updateBuyableDelay*
- *\_updateMinEthBalance* - allows setting 100% discounts.
- *\_updateStandardHashDiscount* - allows setting 100% discounts.
- *\_updateDaoHashDiscount* - allows setting 100% discounts.
- *\_updateEthPercentage* - allows setting 100% discounts. Setting it to 0 will break the *withdrawETH* method.

**Suggested remedy:**

- Add require statements that checks for publicly visible and ideally immutable minimum and maximum values.

4.2 ⚠️ **\_updateBuyableDelay** will affect all previously sold NFT tokens immediately.

**Scenario:**

1. **buyableDelay** set to **2000 blocks**
2. User sells token at block **1000**
3. User expects the token not to be buyable for **2000 blocks**.
4. **TrashBin owner** sets new **buyableDelay** to **1 block**
5. NFT token is available for purchase.

**Suggested remedy:**

- Compute and set buyableDelays per NFT token.
- Restrict min and max buyableDelay for contract owner (see 4.2)

## 5. Potential Exploits

5.1 ! Flooding the TrashBin contract with impostor ERC1155 and ERC721 tokens and thus degrading the usability of the TrashBin token buyers. This exploit becomes more feasible if the contract is deployed on a chain with low gas prices (e.g. Polygon, BSC).

**Exploit PoC Code:**


[https://github.com/CookedCookee/FragmentsDAO/blob/poc-exploit-impostor-nft/contracts/hashes\\_collections/trashbin/ImpostorERC721.sol](https://github.com/CookedCookee/FragmentsDAO/blob/poc-exploit-impostor-nft/contracts/hashes_collections/trashbin/ImpostorERC721.sol)

**Scenario:**

1. Exploiter creates and deploys a smart contract that mimics an ERC721 and ERC1155 contract.
2. Exploiter is able to sell fake NFT tokens to the TrashBin through the following methods:
  1. **sell**
  2. **onERC721Received**
  3. **onERC1155Received**
  4. **onERC1155BatchReceived**
3. With the bootstrapped PoC by PopBlocks, the exploiter will spend more on gas than the amount received for the sold NFT tokens.
4. These fake NFT tokens sold to the TrashBin contract cannot be purchased back by users via the buy method.
5. The TrashBin owner will have to incur gas fees and time to clean up the **nftStorage**.

**Suggested Remedy:**


- The reviewer was not able to identify a viable mechanism that could exclude this exploit.
- An idea could be creating and maintaining a whitelist of sellable NFT contract addresses, though this directly affects the initial value proposition of the business requirements.

5.2  Increasing the **sellPrice** and the **ETH balance** of the **TrashBin** contract can justify exploiters to develop highly gas efficient exploits similar to the one in 5.1 in order to drain the contract of funds by mimicking the sale of NFT tokens.

**Suggested Remedy:**

- Set limits on the **sellPrice** setter (see 4.1)
- Further research on gas efficient exploits that can mimic the sale of fake NFTs.


## 6. Recommendations

6.1  Implement pause, unpause mechanism.

**Description:**

While the pause effect can be achieved by withdrawing all the ETH it would be more elegant to have a paused flag and revert reason for pausing the contract for any reason.

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/Pausable.sol>

6.2  Consider renaming **ethPercentage** variable to a name reflecting that this holds the basis point fee that is transferred to the multi-sig owner.

6.3  Spelling and grammar mistakes were found.

- sold so to
- recieve
- precentage
- NFTs that has been sold
- becuase

# Conclusion

The TrashBin contract aims to solve a viable problem, however its users would be required to investigate the code or documentation in order to understand how they will be remunerated for the tokens they sell to it (see 2.5).

The TrashBin contract is exposed to the risk of being flooded with impostor NFTs that can degrade the usability of the contract and require maintenance. This exploit also has the risk of becoming a fund security risk. Should the sellPrice be set to a value high enough to facilitate highly efficient faking of ERC721 and ERC1155 transfers, the contract could be drained of funds.

It is highly recommended to review the suggested remedies and expand the unit tests to cover their implementation.