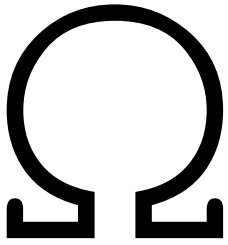
Everbloom Smart Contract Audit

Final Audit Report

February 16, 2023



Team Omega

teamomega.eth.limo

Summary

Everbloom has asked Team Omega to audit the contracts that define the behavior of their NFT contracts.

We found **no high severity issues** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **3** issues as “medium” - this is an issue we believe you should definitely address. In addition, **9** issues were classified as “low”, and **10** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Severity	**Number of issues**	**Number of resolved issues**
High	0	-
Medium	3	3
Low	9	7
Info	10	8

Scope of the Audit

The scope of the audit concerns all solidity files in the following repository:

```
https://github.com/StruccAB/Everbloom-Solidity-Contracts
```

We audited the code at the following commit:

```
c4ff5626996f1aed6a2faa4336c516e6577cdce2
```

The scope of the audit regards the files `contracts` directory, except for the files in the `testContracts` directory

```
contracts
├── EverDropManager.sol
├── EverErrors.sol
├── EverNFT.sol
├── IEverDropManager.sol
├── IEverNFT.sol
```

Resolution

Everbloom subsequently addressed the issues with the final commit of:

```
ad9b3a9b342f6adf51b5879c434b19ec7047e203
```

We reviewed these fixes and updated the issues below.

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. The tools have detected a number of low severity issue that wee have included in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets, or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Code coverage is incomplete [low] [partially resolved]

The code coverage is incomplete and leaves a significant part of the code untested:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	71.28	60.34	78.57	73.02	
EverDropManager.sol	96.88	80.95	93.33	89.13	... 156,292,361
EverErrors.sol	100	100	100	100	
EverNFT.sol	58.06	48.65	61.54	63.75	... 372,373,387
IEverDropManager.sol	100	100	100	100	
IEverNFT.sol	100	100	100	100	
contracts/testContracts/	36.36	25	30.77	40.74	
EverDropManager2.sol	38.1	25	33.33	42.31	... 165,175,187
IEverDropManager2.sol	100	100	100	100	
USDC.sol	0	100	0	0	7
All files	64.66	55.15	63.41	67.32	

Recommendation: Write tests to cover all lines of code. Also instruct the coverage reporter to ignore the `testContracts` directory.
Severity: Low
Resolution: This issue was partially resolved. The coverage was significantly increased and now covers all lines, but a part of the branches remain untested. Also the test files are now properly excluded from the coverage.

G2. initialize function is not automatically called in the implementation contract [low] [resolved]

When deploying the implementation contract of the proxy, it is important to make sure that the implementation contract is initialized - i.e. that the contract's `initialize` function can not be called by a malicious attacker. It is recommended that the deployment and the disabling of the `initialize` function happen in a single transaction, so the initialization can not be frontrun.

Recommendation: call `_disableInitializers()` in the constructor function, e.g. like this

```
constructor() {
      _disableInitializers();
}
```

See also https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable
This applies both to the `EverDropManager` and the `EverNFT` contracts.
Severity: Low
Resolution: This issue was resolved as recommended.

G3. Pin dependencies on solidity packages [info] [resolved]

The `package.json` file specifies a 2 dependencies of solidity code:

```
"dependencies": {
    "@openzeppelin/contracts": "^4.8.0",
    "@openzeppelin/contracts-upgradeable": "^4.8.0",
    …
 }
```

The code from some of these packages is imported in the solidity files, and is part of the compiled result that will be uploaded on-chain. Using fixed dependencies is important to get a predictable result from the compilation, makes your code unambiguous, and will make future verification of the contracts easier.
Recommendation: Pin the version of the OpenZeppelin dependencies that you intend to use (currently, on the basis of `yarn.lock`, this would be:

```
    "@openzeppelin/contracts": "4.8.0",
    "@openzeppelin/contracts-upgradeable": "4.8.0",
```

You could also consider updating to the latest version of 4.8.1, although the fixes offered in the new version do not concern the code used in the contracts.
Severity: Info
Resolution: This issue was resolved as recommended. The versions of the Open Zeppelin contracts dependencies were upgraded to 4.8.1 and pinned.

G4. Event parameters are not consistently indexed [info] [resolved]

The `EverDropManager` and `EverNFT` contracts define multiple events, but the indexing of the parameters of the events is not consistent and seems to be lacking in some cases.
Currently, the only event with indexed parameters is the `NewDrop` event in the `EverDropManager`, and that event includes a duplicate `externalId` parameter: one is indexed, the other one is not. The rest of the events have no indexed parameters.
Recommendation: Indexing parameters can be useful for quick and efficient filtering of events based on the value of the indexed parameter, which is why it is normally recommended to index parameters such

as IDs and addresses, which makes querying of events by front end systems easier. We recommend adding the `indexed` keyword to the `dropId` parameter in all events, and remove the non-indexed `externalId` parameter in the `NewDrop` event.
Severity: Info
Resolution: This issue was resolved as recommended. All `dropId` parameters in the events are now indexed, and the duplicated `externalIdTopic` parameter was removed from the `NewDrop` event.

G5. Interfaces are incomplete [info] [not resolved]

The `IEverDropManager` and `IEverNFT` interface does not define all functions in the `EverDropManager` and `EverNFT` - for example the `IEverNFT` does not define or inherit any of the `ERC721` methods. This makes the interface not very useful for external uses.
Recommendation: Define complete interfaces for the `EverDropManager` and `EverNFT` contracts that expose all the methods that your users might use.
Severity: Info
Resolution: This issue was not resolved.

EverDropManager.sol

EDM1. Improper use of the AccessControl role management functionality [low] [resolved]

The `AccessControlUpgradeable` contract that the contract inherits from defines the public `grantRole` and `revokeRole` functions, which allow the admin of a role (by default the `DEFAULT_ADMIN_ROLE`) to grant and revoke roles. However, the contract also defines its own logic for granting and revoking roles, with the `addSubAdmin`, `removeSubAdmin`, `addCreator`, and `removeCreator`.
This is unnecessary, and can cause unexpected behavior of the role management permissions. For example, the `addCreator` and `removeCreator` functions state that “Only the contract `SUB_ADMIN_ROLE` can perform this operation”, but in fact, also the `DEFAULT_ADMIN_ROLE` can perform it directly by calling the `grantRole` and `revokeRole` functions.
Recommendation: Remove the `addSubAdmin`, `removeSubAdmin`, `addCreator`, and `removeCreator` functions and in the `initialize` function call `_setRoleAdmin` to set the `SUB_ADMIN_ROLE` as the admin role of the `CREATOR_ROLE`.
Severity: Low
Resolution: This issue was resolved as recommended.

EDM2. EverDropManager should inherit from IEverDropManager [low] [resolved]

The `EverDropManager` contract should inherit from `IEverDropManager` so that the compiler can check that the interface is defined correctly. Also the definition of structs are currently duplicated in the two files, which could be avoided by inheriting the interface (or at least importing the structs from it).
Recommendation: Make the `EverDropManager` contract inherit from `IEverDropManager`, and move the definition of structs and events to be in the interface.
Severity: Low
Resolution: This issue was resolved as recommended.

EDM3. Use of deprecated method _setupRole [info] [resolved]

In line 99 the `_setupRole` function is used, but it was deprecated in favor of the `_grantRole` method (which is correctly used in the lines below).
Recommendation: Replace the call to `_setupRole` with `_grantRole`.
Severity: Info
Resolution: This issue was resolved as recommended.

EDM4. initialize function should call __ERC165Storage_init [info] [resolved]

The contract calls all __Foo__init functions for each inherited contract Foo that implements such a function, except for `__ERC165Storage_init`. Although these functions currently do nothing, they provide an unambiguous upgrade path for the OpenZeppelin dependencies.
Recommendation: Also call `__ERC165Storage_init` in the `initialize` function.
Severity: Info
Resolution: This issue was resolved as recommended.

EDM5. getDrop function is not needed [info] [not resolved]

Because the `drops` array is declared public, the solidity compiler will automatically create a getter function `drops(uint)`, and so defining an explicit getter function is not needed.
Recommendation: Remove the `getDrop()` function from the contract.
Severity: Info
Resolution: This issue was not resolved.

EverNFT.sol

EN1. Minting does not verify that transfer of tokens was successful [medium] [resolved]

In case the drop's price is not 0, the contract will try to transfer tokens from the minter to the treasury using the following code:

```
ERC20(erc20tokenAddress).transferFrom(msg.sender, treasury,
drop.tokenInfo.price * _quantity);
```

This code does not verify that the transfer of tokens was successful: the ERC20 token standard only requires that the transferFrom function returns true on successful transfer, it does not require that the function throws an error.
To make sure that the payment was successful,, the contract must check that the call to `transferFrom` returned true. Without checking the return value, tokens that return `false` (instead of reverting) on a failed transfer could make the minting pass even if the payment for it has failed.
Recommendation: Use OpenZeppelin's SafeERC20's `safeTransferFrom`, or require that the `transferFrom` function return true.
Severity: Medium
Resolution: This issue was resolved as recommended. OpenZeppelin's SafeERC20's `safeTransferFrom` is now used.

EN2. Payment token might change without user agreement [medium] [resolved]

The token with which the payment for minting is made can be changed at any moment by the owner with a call to `setERC20TokenAddress`.
In the case where a user has an existing approval of another token to the contract, this could cause a situation where the user intends to mint for a certain amount of token A, sends a transaction calling to mint, then a transaction changing the payment token to token B is made and included in a block first, and then the user's mint transaction occurs. The result is that the user makes a payment in a token the user didn't expect to pay with.
Recommendation: Unless there are strong reasons to change the token in which NFTs are paid for, remove the option to change the token that is used for paying for minting.
Severity: Medium
Resolution: This issue was resolved as recommended. The token used for payment can no longer be changed.

EN3. Minting can be prevented by others [medium] [resolved]

Since transactions that try to mint more than the total supply of the NFT will revert, it is possible for an attacker to make the transactions of others revert by minting himself the smallest amount that, along with another user's mint transaction, will make the other's mint more than the total supply. For example, if there are 1000 NFTs available, and a user is now sending a transaction to purchase all of them, an attacker could front run their transaction and buy just 1 NFT, which will make the original buyer's transaction to revert. This may even happen in the normal flow of the system, but an attacker could do this repeatedly to cause disruptions in the system.
Recommendation: Allow users to specify if they wish to mint the maximum amount of NFTs available, assuming the quantity they have specified is greater than the maximum available.
Severity: Medium
Resolution: This issue was resolved as recommended.

EN4. Follow the checks-effects-interaction pattern in the mint() function [low] [resolved]

The `mint` function uses the `_safeMint` function to mint the NFT. The `_safeMint` function makes an external call to the receiver of the NFT, which may be an unknown contract. The call to `_safeMint` is done before the state changes are made, as the `externalIdToTokenId` mapping is updated after the call. This opens the possibility for reentrancy, which could allow calling the `mint` function twice with the same `externalId`.
Recommendation: Make state changes before external calls. In this case, you should call `_safeMint` only after updating the `externalIdToTokenId` mapping.
Severity: Low
Resolution: This issue was resolved as recommended. Also, the `externalIdToTokenId` mapping was removed. The checks-effects-interaction pattern is not completely implemented: it is possible that the receiver contract re-enters - but we do not think there is the possibility of an attack here.

EN5. tokensOfOwner may run out of gas [low] [resolved]

The `tokensOfOwner` function contains an unbounded loop over the tokens of a user, which may run out of gas if the user has too many tokens.
Recommendation: If this method is only needed for off chain use, consider removing the function and moving the logic off chain.
Severity: Low
Resolution: This issue was resolved as recommended. The `tokensOfOwner` function was removed.

EN6. getIneligibilityReason is unnecessary and misses a condition check [low] [partially resolved]

The `getIneligibilityReason` imitates the `mint` function - it checks for the ability to mint with certain parameters, and returns a string explaining the error if minting would fail.
There are two problems with this implementation. First of all, the behavior of the two functions does not exactly correspond: for example, if the contract is paused, minting will fail, but the `getIneligibilityReason` will not explain why.
There is an easier way to obtain the reason for failure of execution of a function in solidity - most ethereum nodes (and web3 clients) support the `eth_call` method, which will simulate the execution and return the error if an error is thrown - implementing a separate function is not necessary.
Recommendation: Remove the `getIneligibilityReason` function to avoid the unnecessary code duplication or, if you decide to keep it, add a check for the case that the contract is paused.
Severity: Low
Resolution: This issue was partially resolved. The `getIneligibilityReason` function still exists, but it now also checks that the contract is not paused. However, the `getIneligibilityReason` function may still miss other failure reasons, for example if the caller doesn't have enough tokens or allowance to make the payment for the minting.

EN7. EverNFT should inherit from IEverNFT [low] [resolved]

The `EverNFT` contract should inherit from the `IEverNFT` interface, so that the compiler can check that the interface is defined correctly.
Recommendation: Make the `EverNFT` contract inherit from `IEverNFT`, and move the definition of events to be in the interface.
Severity: Low
Resolution: This issue was resolved as recommended.

EN8. mint function can be front-run to block a user to mint certain external IDs [low] [resolved]

The `mint` function takes a list of external IDs which are then connected to the minted NFTs. An external id can not be reused. This means that an attacker can front-run the mint transaction sent by a user and block the transaction by minting one of the NFTs on the list.
An attacker may do this because they may believe that the fact that a user uses a certain external ID is a signal that the NFT is more valuable than others.

Recommendation: If the external ID can be any number specified by the user (as it currently is), there is no easy solution to this problem.
Severity: Info
Resolution: This issue was resolved. External IDs for tokens were removed from the code.

EN9. Manage token and price in the same location [info] [resolved]

The price of an NFT is given by its token, which is set in the `EverNFT.erc20tokenAddress` and the `price`, which is stored in the `EverDropManager` contract. This is confusing.
Recommendation: Define the price and the token to be defined in a single place (either the NFT contract or the Manager contract) instead of managing it in two separate locations.
Severity: Info
Resolution: This issue was resolved as recommended. The token definition was moved to the `TokenInfo` struct to be defined together with the price.

EN10. initialize function should call all inherited __init methods [info] [resolved]

The contract calls all __Foo__init functions for each inherited contract Foo that implements such a function, except for `__ERC721Enumerable_init` and `__ERC721Pausable_init`. Although these functions currently do nothing, they provide an unambiguous upgrade path for the OpenZeppelin dependencies.
Recommendation: Also call `__ERC721Enumerable_init` and `__ERC721Pausable_init`. in the `initialize` function.
Resolution: This issue was resolved as recommended.

EN11. Consider using externalIds as tokenIds [info] [resolved]

The code can be considerably simplified and made more gas efficient by using the provided `externalId` in the `mint` function as the `tokenId`.
The current implementation manages a mapping `externalIdToTokenId` and implements logic to make sure an external id is unique within the NFT contract, and that each `tokenId` is unique as well. This storage and logic can be considerably simplified by taking the provided `externalId` as the `tokenId` of the new NTF.
Recommendation: If newly generated consecutive tokenIds are not needed, then consider to implement this simplification.
Severity: Info
Resolution: This issue was resolved. External IDs for tokens were removed from the code.

EN12. Fix linter warning [info] [resolved]

When compiling, the linter currently gives a warning that the `_baseURI` parameter of the `initialize` function shadows the declaration of the `_baseURI` function. This can be resolved by renaming the `_baseURI` parameter to have a unique name.
Recommendation: Fix the linter warning and avoid the naming duplication by renaming the `_baseURI` parameter of the `initialize` function.
Severity: Low
Resolution: This issue was resolved as recommended.