macro

thirdweb A-13

Security Audit

June 30th, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for thirdweb's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from June 13, 2023 to June 23, 2023.

The purpose of this audit is to review the source code of certain thirdweb Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	1	-	-	1
Medium	1	-	-	1
Code Quality	4	-	-	4
Informational	2	-	-	2

thirdweb was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Slack with the thirdweb team.
- A audit handoff document provided through Notion.

Source Code

The following source code was reviewed during the audit:

- **Repository:** contracts

  Commit Hash **(Dynamic Drops):** `3cb172774ecbc4a77b7713ddd0e1ca6d56eb412c`

  Commit Hash **(Loyalty Card):** `92dca09d6e621f74eb2462b8f338a15886875917`

We audited the following contracts specificially for the Dynamic Drops audit:

Contract	SHA256
contracts/drop/DropERC1155.sol	`0221834d3c7730d145930f479fc98bcc03616eabfe23c497b14eb9d1b2c86dd9`
contracts/drop/DropERC20.sol	`0a28be6097dcd800e22e100c782f45de8cbe812705f4709e9f292ad1a30697b0`
contracts/drop/DropERC721.sol	`58416dca2883e77353d01b9c4260e4a9f574a996e3dc64f374c9d894aac2a557`
contracts/drop/extension/DropERC1155Logic.sol	`1f0e424580d57138ec83673e8f0bf36ed4acbdcc611d29a13eef57e08190a5fe`
contracts/drop/extension/DropERC1155Storage.sol	`a464054c1c1739f2b0a00cc3199d2387e741fb561e04625cd8b0755b1e35ac91`
contracts/drop/extension/DropERC20Logic.sol	`0b60e52a768ed82f904f37dfcd2199679185a37285821d6e039fd4b110c216aa`
contracts/drop/extension/DropERC20Storage.sol	`2e4d2b99fcc3634bd7301546e98f92651981f3e97c2696a0fd9f5c0a9cffcbd6`
contracts/drop/extension/DropERC721Logic.sol	`7bfbe8b00f98f5ef7e4e1079dbb9d088d466801c50c7279973119b3d5eb38f24`
contracts/drop/extension/DropERC721Storage.sol	`fe7442085abda7c6599e9dd5d80300a8d38b977dd3e1a5b006366bf2b4df82d7`

We audited the following contracts specificially for the Loyalty Card audit:

Contract	SHA256
contracts/LoyaltyCard.sol	964f38e8692c8d932cd1d7be3494ffa04e6bdcf349b7ac9d8cf0dfd348f864b7

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
   - How bad things can get (for a vulnerability)
   - The significance of an improvement (for a code quality issue)
   - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
   - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ Mint price is determined by quantity

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	High	Medium

`LoyaltyCard.mintWithSignature` always only mints 1 NFT to the user as stated in the comments:

```
/// @dev Mints an NFT according to the provided mint request. Always mints 1 NFT.
```

and defined by the line LoyaltyCard#L276:

```
_safeMint(_to, 1);
```

However, in `_collectPrice`, the total price the user has to pay is calculated as follows:

```
uint256 totalPrice = _quantityToClaim * _pricePerToken;
```

Thus, for quantity > 1, the user has to pay the total price determined by the quantity, despite only getting 1 NFT minted.

Remediations to Consider

Consider only allowing mint request with quantity = 1 and otherwise revert.

~~M-1~~ Native token can get locked in drop contracts

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	High	Low

Reference: DropERC20Logic.sol#L124, DropERC721Logic.sol#L199, DropERC1155Logic.sol#L206

Description

`DropERC20Logic` inherits from `Drop` contract which implements the `payable` `claim(…)` function that calls `_collectPriceOnClaim` to transfer payment to `feeRecipient` and `saleRecipient` .

The issue of locking native tokens in the contract occurs under the following circumstances:

1. Payment currency in `claimConditions` is set to ERC20.
2. A user calls `claim` with appropriate amount of ERC20 tokens approved and accidentally also passes native tokens along.

As a result, the `claim` call succeeds but the native tokens passed along will be locked within the DropERC20 contract.

Note that this issue also applies to `DropERC721Logic` and `DropERC1155Logic` .

An admin could rescue the tokens by adding a new extension that provides appropriate function to transfer tokens back to original owners. However, it is recommended to avoid such situations in the first place.

Remediations to Consider

Consider checking that no native tokens are transferred (`msg.value == 0`) when payment currency is set to ERC20.

~~Q-1~~ `ReentrancyGuardUpgradeable` not initialized

TOPIC	STATUS	QUALITY IMPACT

Best Practice	Fixed	Low

`LoyaltyCard` inherits from `ReentrancyGuardUpgradeable` but is not being initialized in the `initialize` function.

Note that due to the logic in `ReentrancyGuardUpgradeable`, this doesn’t impose any security risk, but it is considered as best practice to properly initialize all parent contracts.

Remediations to Consider

Consider initializing `ReentrancyGuardUpgradeable` properly by adding `__ReentrancyGuard_init` to the `initialize` function.

~~Q-2~~ Unused role `OPERATOR_ROLE`

TOPIC	STATUS	QUALITY IMPACT
Clean Code	Fixed	Low

`OPERATOR_ROLE` is declared in the following contracts:

- DropERC721.sol#L83
- DropERC1155.sol#L88
- DropERC721Logic.sol#L66
- DropERC1155Logic.sol#L64

However, this role is not used anywhere else in the code

Remediations to Consider

Consider removing `OPERATOR_ROLE` declarations from above contracts.

~~Q-3~~ Unused Import

TOPIC	STATUS	QUALITY IMPACT
Clean Code	Fixed	Low

The following imports of `ReentrancyGuardInit` are not required and can be removed:

- DropERC20.sol#L27
- DropERC721.sol#L31
- DropERC1155.sol#L32

The following import of `ERC20VotesStorage` is not required and can be removed:

- DropERC20Logic.sol#L21

~~Q-4~~ Storage gap variable not needed

TOPIC	STATUS	QUALITY IMPACT
Clean Code	Fixed	Low

Due to the use of dynamic contract pattern and its use of unstructured storage, `__gap` variables declared in some of the upgradable contracts are not needed. Consider removing the `__gap` variable from the following files:

- ERC2771ContextUpgradeable.sol#L66
- ERC20BurnableUpgradeable.sol#L50
- ERC20Upgradeable.sol#L393
- ERC20VotesUpgradeable.sol#L276
- draft-EIP712Upgradeable.sol#L122

- draft-ERC20PermitUpgradeable.sol#L107

~~I-1~~ Transfers are enabled by default

TOPIC	STATUS	IMPACT
Use Case	Fixed ↗	Informational ✳

`LoyaltyCard` allows the transfers of NFTs by default; enabled by the following line in `initialize`:

```
_setupRole(TRANSFER_ROLE, address(0));
```

Due to the use case of `LoyaltyCard` - namely issuing NFTs to loyal customers - consider restricting transfers by default for normal users and only allow transfers for admins having the `TRANSFER_ROLE` assigned.

~~I-2~~ No support for flat platform fee

TOPIC	STATUS	IMPACT
Use Case	Fixed ↗	Informational ✳

`OpenEditionERC721` has been recently updated to support a new fee mechanism called "flat fee" besides "percentage fee". However, all three drop contracts only support "percentage fee". Consider updating drop contracts to support "flat fee" in addition to "percentage fee".

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.