macro

thirdweb A-5

Security Audit

November 11th, 2022
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 September 26, 2022 to October 7, 2022.

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
Medium	1	-	-	1
Code Quality	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 provided audit handoff document provided through Notion.

Source Code

The following source code was reviewed during the audit:

- **Repository:** contracts
- **Commit Hash:**

Specifically, we audited the following contracts:

Contract	SHA256
contracts/drop/DropERC20.sol	`095c5b160446f675c3832f9166294639cb389ceef74be3d51a4bc8339c46423f`
contracts/drop/DropERC721.sol	`cc9d276ed99ce1d54755937c449dddcc28a40872aa2f8ad4fea8b54fafa83c28`
contracts/drop/DropERC1155.sol	`2272277a6d1b0f41da26ffc32313adba43ad3f7fb691371a3fba241f28edfb8b`

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.

- ~~M-1~~ Merkle claiming and normal claiming occupy the same state space
- Q-1 721a Transfer Costs
- Q-2 Ability to set individual claim count

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

~~M-1~~ Merkle claiming and normal claiming occupy the same state space

TOPIC	STATUS	IMPACT	LIKELIHOOD
Data Model	Addressed	Medium	Medium

Documented the limitations of new design.

When claiming, ClaimCondition's `quantityLimitPerWallet` is validated and incremented. This also happens for Merkle proof claiming, a situation where:

- An admin creates a merkle node with a specified limit, currency, and price per token for a specific address
- The admin gives the proof of said node to a user
- The user mints using said proof, "overriding" the default settings of the current claim condition.

This all works as advertised. However, **both** normal claims *and* Merkle proof claims validate and increment `quantityLimitPerWallet`. This means the admin and user are vulnerable to subtle usage pitfalls. For example, take the following scenario:

- Admin creates an NFT drop where each token costs 1 WETH to mint.
- Admin wishes to grant user X permission to mint 2 tokens at 0.25 WETH, and 4 tokens at 0.5 WETH.

In this scenario, several issues may occur:

1. Knowing user X should be able to mint 2 + 4 tokens, admin adds two merkle leaves with `quantityLimitPerWallet` set to 6. However, this allows user X to mint 6 tokens at 0.25 WETH. To do this properly, the admin must update the merkle tree twice – the second time *only after* user X has fully minted the first.
2. Knowing user X should only be able to mint 2 and 4 tokens for 0.25 WETH and 0.5 WETH respectively, admin sets one leaf's `quantityLimitPerWallet` to be 2, and the other to be 4. However, this only allows user X to mint 4 tokens total (instead of 6).
3. Knowing user X needs to mint one price at a time, admin sets *one* leaf's `quantityLimitPerWallet` to be 2 for 0.25 WETH. However, in the meantime, user X has minted one token at the normal price; they can now only claim 1 token at the discounted price.

Other contracts / software systems that maintain merkle trees for these drop contracts might also create more severe vulnerabilities for themselves.

To help avoid the above confusions, consider:

1. Adding an optional namespace key in `struct AllowlistProof`, to allow limitations per topic (e.g. a specific currency/price) instead of just per address; or
2. Documenting the limitations & pitfalls of the current implementation.

Q-1 721a Transfer Costs

TOPIC	STATUS	QUALITY IMPACT
Use Cases	Wont Do	Low

Transferring a token from a large mint batch may cost a very high amount of gas.

Consider implementing an optional max batch size, and consider including / referring to docs about how to optimally transfer minted tokens (example).

RESPONSE BY THIRDWEB

> Not fixing.
>
> Moving to ERC721-A for all NFT Drops.

Q-2 Ability to set individual claim count

TOPIC	STATUS	QUALITY IMPACT
Use Cases	Wont Do	Low

In the previous version of the contracts, a `setWalletClaimCount()` function was available for an admin to manually set an addresses's claim count.

In the current version, no such function exists.

Consider implementing a similar function, if it is desirable for an admin to be able to manually update these values without having to deal with merkle proofs.

RESPONSE BY THIRDWEB

> Not fixing right now.

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.