macro

thirdweb A-15

Security Audit

October 6th, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Disclaimer

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 11, 2023 to September 22, 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
Medium	1	-	-	1
Low	3	-	-	3
Code Quality	12	-	-	12
Gas Optimization	1	-	-	1

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.

Trust Model, Assumptions, and Accepted Risks (TMAAR)

MarketplaceV3 and **BurnToClaimDropERC721** contracts implement role-based access control including the roles listed below. In particular, DEFAULT_ADMIN_ROLE and EXTENSION_ROLE have high privileges such as making upgrades to the contracts. Both roles are assumed to be trusted and to act in a reliable and good manner.

Specifically, the different roles have the following privileges:

MarketplaceV3

`DEFAULT_ADMIN_ROLE` : grant and revoke roles; set platform fee info; set contract URI; set royalty engine

`EXTENSION_ROLE` : add, replace, or remove extensions

`LISTER_ROLE` : create listings and auctions. Only applies when restriction is enabled. By default, restriction is disabled so that everybody can create listings and auctions.

`ASSET_ROLE` : Only NFT contracts with ASSET_ROLE can be listed or auctioned, when restriction is enabled. By default, restriction is disabled so that every NFT can be listed or auctioned.

BurnToClaimDropERC721

`DEFAULT_ADMIN_ROLE` : grant and revoke roles; set platform fee info; set contract URI; set royalty info; set primary sale recipient; set owner; set claim conditions

`EXTENSION_ROLE` : add, replace, or remove extensions

`MINTER_ROLE` : lazy mint tokens; reveal the URI

`TRANSFER_ROLE` : transfers to or from `TRANSFER_ROLE` holders are valid, when transfers are restricted.

Source Code

The following source code was reviewed during the audit:

- **Repository:** contracts
- **Commit Hash:** `46e69070978c23b9533edb381e838a5dddf7ed9d`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
contracts/prebuilts/marketplace/IMarketplace.sol	`3313a1a020c6dea590f54b0be51da0746f6ae403f237de4b331ed8d3c2ba00cc`
contracts/prebuilts/marketplace/direct-listings/DirectListingsLogic.sol	`34eefa1719ac1ac314406d59c6b0c8a54a6db5ff7ca3b16c16281f0ae23c5a0d`
contracts/prebuilts/marketplace/direct-listings/DirectListingsStorage.sol	`323b76b4ac66470241fd83b76d89882edf0514eafbb8f3a3e507f798865f8fb7`
contracts/prebuilts/marketplace/english-auctions/EnglishAuctionsLogic.sol	`5c3e0e3d331c38b1f9487230002f0f68b8972d10232ad435338229549995f52e`
contracts/prebuilts/marketplace/english-auctions/EnglishAuctionsStorage.sol	`d6c11797d33d215b5101c5d68884df8038e6ab94f7873847a001484b52091fa4`
contracts/prebuilts/marketplace/entrypoint/MarketplaceV3.sol	`ee794f6bdb80442fc34cfc0cd646e9d7ccdbc345bc52f15a814b1bdd52041a93`
contracts/prebuilts/marketplace/offers/OffersLogic.sol	`3e00e6ccfd4a7b93c3648628ee16fce2132b88815577fd2db6bca9b3e63bbc05`
contracts/prebuilts/marketplace/offers/OffersStorage.sol	`7eaa2bb7340ebe38bfbed618fca403cd55435964b7d0742748d155aed17865f2`

Contract	SHA256
contracts/prebuilts/unaudited/burn-to-claim-drop/BurnToClaimDropERC721.sol	1da3f3c6359815ff480629fe879143b94d43a5cf0b7bbcde323ba3691e82ff9e
contracts/prebuilts/unaudited/burn-to-claim-drop/extension/BurnToClaimDrop721Logic.sol	7db94a6af68b73a4fbf7d0af53b4a1f3e3483a569e1765456f3189fa85038096
contracts/prebuilts/unaudited/burn-to-claim-drop/extension/BurnToClaimDrop721Storage.sol	2094fddca9d0b4018e8d66bc07dda4ccd5ffc1d120ed6134597f3929ad614900

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.

In addition, for the marketplace contracts, the audit was focused on the changes made since the previous audit of MarketplaceV3 including:

- latest plugin pattern setup
- support for Manifold's Royalty Registry

Issue Descriptions and Recommendations

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

- ~~M-1~~ Native tokens can get locked when bidding or buying from the marketplace
- ~~L-1~~ Everyone can send native tokens directly to the marketplace
- ~~L-2~~ Missing sanity checks for `PrimarySale` and `PlatformFee` recipients
- ~~L-3~~ Cannot remove upgradability without revoking all default admins
- ~~Q-1~~ Misleading function name for `setMaxTotalSupply`
- ~~Q-2~~ Missing sanity check when setting `BurnToClaimInfo`
- ~~Q-3~~ `OPERATOR_ROLE` is not used
- ~~Q-4~~ Missing `natspec` documentation
- ~~Q-5~~ Named returns are not assigned
- ~~Q-6~~ Avoid code duplication by using `maxTotalSupply()`
- ~~Q-7~~ Adhere to naming convention for internal function
- ~~Q-8~~ Incorrect `VERSION` used
- ~~Q-9~~ Document behavior of `tokenURI`
- ~~Q-10~~ Use `ERC721Holder` and `ERC1155Holder`
- ~~Q-11~~ Avoid code duplication by using `_msgSender` and `_msgData`
- ~~Q-12~~ Comply with `ERC7201` standard
- ~~G-1~~ Reorder struct members to save storage slots

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~~ Native tokens can get locked when bidding or buying from the marketplace

TOPIC	STATUS	IMPACT	LIKELIHOOD
Locked Funds	Fixed	Medium	Medium

`EnglishAuctionsLogic`'s `bidInAuction` function is marked as `payable` to accept native tokens for bids. However, for auctions using ERC20 as currency, when a user unintentionally sends `msg.value > 0` when calling `bidInAuction`, the transaction will succeed and the native tokens will be locked in the contract.

The tokens will be locked in the contract, unless a permissioned user adds an extension to enable withdrawing these tokens. In cases where extension permissions have been revoked, there would be no way to withdraw these tokens.

Note that the issue also applies to `DirectListingsLogic's buyFromListing` function.

Remediations to Consider

Consider adding a check to verify that `msg.value == 0` when currency is set to ERC20.

~~L-1~~ Everyone can send native tokens directly to the marketplace

TOPIC	STATUS	IMPACT	LIKELIHOOD
Locked Funds		Medium	Low

Fixed

Fixed

`MarketplaceV3` supports sending native tokens directly to the contract via the `receive()` function. It is understood that this is needed in order to support WETH for listings.

Since there is no restriction on who can call `receive()`, native tokens can get locked when users accidentally send them to the contract.

Remediations to Consider

Consider adding a restriction to the `receive()` function so that it is callable only from the `nativeTokenWrapper` address. An example of restricting the `receive()` function to the WETH address can be found in the UniswapRouter contract.

~~L-2~~ Missing sanity checks for `PrimarySale` and `PlatformFee` recipients

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Validation	Fixed	Medium	Low

Fixed

`PrimarySale` and `PlatformFee` recipients can be set to 0x0 during initialization via `BurntToClaimDropERC721.initialize` or via `setPrimarySaleRecipient` and `setPlatformFeeInfo` respectively.

If one of those recipients is set to a 0x0 and corresponding fee > 0 this can result in the following behavior:

- For claims using ERC20 as currency, claiming a token will revert in `_collectPriceOnClaim` with *"ERC20: transfer to the zero address"*

- For claims using native token as currency, tokens will be lost as they are transferred to the 0x0 address.

Note that this issue applies to all contracts supporting `PrimarySale` and `PlatformFee`.

Remediations to Consider

Consider adding checks to prevent setting `address(0)` for both `PrimarySale` and `PlatformFee` recipients.

~~L-3~~ Cannot remove upgradability without revoking all default admins

TOPIC	STATUS	IMPACT	LIKELIHOOD
Upgradability	Fixed	Low	Low

In `MarketplaceV3` the ability to add or update extensions to the contract can be done by an account with the `EXTENSION_ROLE`. This role can only be granted and revoked by any account with the `DEFAULT_ADMIN_ROLE`, since there is no role admin set for the `EXTENSION_ROLE`. In the case where a project using these contracts wants to turn off the ability to add/update extensions, they would have to revoke all users with the `EXTENSION_ROLE` as well as users with the `DEFAULT_ADMIN_ROLE`, since they can grant the `EXTENSION_ROLE` to another user at a later time.

Revoking all accounts with the `DEFAULT_ADMIN_ROLE` may be undesirable as it also manages other roles like the `LISTER_ROLE` and `ASSET_ROLE`.

Remediations to Consider

Set the `EXTENSION_ROLE` as its own role admin in the initializer and set an initial account with the `EXTENSION_ROLE`, this will allow it so the contract can no longer be upgraded when there are no accounts with the `EXTENSION_ROLE`.

~~Q-1~~ Misleading function name for `setMaxTotalSupply`

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed Fixed Fixed	Low

The intention of `BurnToClaimDrop721Logic's setMaxTotalSupply` function is to set the **maximum number of tokens to be minted** rather than - as the name implies - setting the maximum number of token supply.

Remediations to Consider

Consider renaming the function `setMaxTotalSupply` and the state var `maxTotalSupply` to something more appropriate such as `setMaxTotalMinted` and `maxTotalMinted`.

~~Q-2~~ Missing sanity check when setting `BurnToClaimInfo`

TOPIC	STATUS	QUALITY IMPACT
Input Validation	Fixed Fixed	Low

In BurnToClaim.sol, an admin can set `BurnToClaimInfo` via `setBurnToClaimInfo(..)`. However, both `originContractAddress` and `currency` parameter can be set to `0x0` as there is no check preventing this. As a result, subsequent calls to `burnAndClaim` would revert inside the `verifyBurnToClaim` function.

Remediations to Consider

It is recommended to prevent setting invalid config in the first place, thus consider adding `!= address(0)` checks to the `setBurnToClaimInfo` function and remove the check in the `verifyBurnToClaim` function.

~~Q-3~~ `OPERATOR_ROLE` is not used

TOPIC	STATUS	QUALITY IMPACT
Extra Code	Fixed	Low

In `BurnToClaimDropERC721` and `BurnToClaimDrop721Logic`, the `OPERATOR_ROLE` is defined but not used anywhere in the code.

Remediations to Consider

Consider removing `OPERATOR_ROLE` definitions and corresponding `_setupRole` logic from above contracts.

~~Q-4~~ Missing `natspec` documentation

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	Low

`BurnToClaim.sol` misses proper `natspec` documentation for most of the functions. In some of the other contracts such as `BurnToClaimDropERC721.sol`, `BurnToClaimDrop721Logic.sol`, and `BurnToClaimDrop721Storage.sol`, most of the functions take use of the `@dev` tag, but they tend to not include `@param` and `@return` tags.

Remediations to Consider

Add missing `natspec` documentation. Follow natspec guidelines to provide proper documentation of the code.

~~Q-5~~ Named returns are not assigned

TOPIC	STATUS	QUALITY IMPACT
Code Readability	Fixed	Low

In `BurnToClaimDrop721Logic`, there are two ocurrences where named returns are defined but not assigned:

1. The function `lazyMint` defines a named return variable `batchId`, but the value is returned directly instead of assigning it to the variable.
2. The function `_msgSender` defines a named return variable `sender`, but the value is returned directly instead of assigning it to the variable.

Remediations to Consider

Consider either assigning the return value to the defined return variable or remove the variable at all.

~~Q-6~~ Avoid code duplication by using `maxTotalSupply()`

TOPIC	STATUS	QUALITY IMPACT
Code Readability	Fixed	Low

In `BurnToClaimDrop721Logic`, the function `_checkTokenSupply` retrieves the state var via data storage but could use `maxTotalSupply()` instead to improve code readability.

Remediations to Consider

Consider using `maxTotalSupply()` to reduce code size and improve readability.

~~Q-7~~ Adhere to naming convention for internal function

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed Fixed	Low

In `BurnToClaimDropERC721`, the function `isAuthorizedCallToUpgrade` is an internal function but not prefixed with an underscore.

According to Solidity naming conventions - and as applied everywhere else in the code - private and internal functions should be prefixed with an underscore.

Remediations to Consider

Consider renaming the function from `isAuthorizedCallToUpgrade` to `_isAuthorizedCallToUpgrade`.

~~Q-8~~ Incorrect `VERSION` used

TOPIC	STATUS	QUALITY IMPACT
Upgradability	Fixed	Low

In `MarketplaceV3.sol` the constant `VERSION` is set to `1`, but version `2` was already used in a previous version of the Marketplace contracts.

Remediations to Consider

Consider increasing the `VERSION` to `3`.

~~Q-9~~ Document behavior of `tokenURI`

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	Medium

`BurnToClaimDrop721Logic's tokenURI` function returns a valid URI once the tokenId is lazy minted. However, the tokens are technically not minted until they are claimed. Also for burned tokens the function will return a valid URI.

Although above behavior is as designed, it doesn't fully comply to the ERC721 standard, which says that tokenURI should throw an exception for invalid tokenIds.

Remediations to Consider

Consider adding documentation to make users aware that above behavior slightly deviates from ERC721 standard.

~~Q-10~~ Use `ERC721Holder` and `ERC1155Holder`

TOPIC	STATUS	QUALITY IMPACT
Code Readability	Fixed	Low

In `MarketplaceV3`, the functions `onERC721Received`, `onERC1155Received`, and `onERC1155BatchReceived` are implemented to indicate support of retrieving ERC721 and ERC1155 tokens.

Remediations to Consider

Consider deriving from OpenZeppelin's ERC721Holder and ERC1155Holder instead to improve readability.

~~Q-11~~ Avoid code duplication by using `_msgSender` and `_msgData`

TOPIC	STATUS	QUALITY IMPACT
Code Readability	Fixed	Low

In `MarketplaceV3`, `_msgSender` and `_msgData` are overridden from `ERC2771ContextUpgradable` and `Permission` and the ident function logic as from `ERC2771ContextUpgradable` is reimplemented again.

Remediations to Consider

Consider re-using the logic from `ERC2771ContextUpgradable` by calling `ERC2771ContextUpgradeable._msgSender()` and `ERC2771ContextUpgradeable._msgData()` respectively to improve readability and reduce code size.

~~Q-12~~ Comply with `ERC7201` standard

TOPIC	STATUS	QUALITY IMPACT
Upgradability		Medium

Fixed

Fixed

Thirdweb uses the “namespaced storage” (aka “storage struct”) pattern for all upgradable contracts. However, those upgradable contracts are currently not adhering to the ERC7201 standard, which standardizes the storage location used for the “namespace”.

According to ERC7201, this is important because:

> *These storage usage patterns are invisible to the Solidity and Vyper compilers because they are not represented as Solidity state variables. Smart contract tools like static analyzers or blockchain explorers often need to know the storage location of contract data. Standardizing the location for storage layouts will allow these tools to correctly interpret contracts where these design patterns are used.*

A great example of adhering to the ERC7201 standard can be seen in the recently released OpenZeppelin’s v5 pre-release contracts, e.g. see Initializable.sol.

Remediations to Consider

Consider changing the upgradable contracts to adhere to the ERC7201 standard.

~~G-1~~ Reorder struct members to save storage slots

TOPIC	STATUS	GAS SAVINGS
Gas Optimizations	Fixed	Medium

In `IMarketplace.sol`, the struct members of `Listing`, `Auction`, and `Offer` can be reordered to save storage slots.

In particular, when the struct members `TokenType` (1 Byte), `Status` (1 Byte), and one of the `address` members (20 Bytes) are placed next to each other, they only take 1 storage slot.

Consider reordering the `Listing` struct from:

```
struct Listing {
    uint256 listingId;
    address listingCreator;
    address assetContract;
    uint256 tokenId;
    uint256 quantity;
    address currency;
    uint256 pricePerToken;
    uint128 startTimestamp;
    uint128 endTimestamp;
    bool reserved;
    TokenType tokenType;
    Status status;
}
```

to:

```
struct Listing {
    uint256 listingId;
    uint256 tokenId;
    uint256 quantity;
    uint256 pricePerToken;
    uint128 startTimestamp;
    uint128 endTimestamp;
    address listingCreator;
    address assetContract;
    address currency;
    TokenType tokenType;
    Status status;
    bool reserved;
}
```

The same rules apply to the `Auction` and `Offer` struct.

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 thirdweb 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.