Thirdweb A-2

Security Audit

June 24th, 2022

Version 1.0.0

Prepared by

0xMacro.com

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 1, 2022 to June 17, 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

We identified a few issues of non-severe to high severity. thirdweb was quick to respond and fix these issues.

Specification

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

- Discussions on Slack with the thirdweb team.
- The official website, developer documentation and more specifically provided documentation for contracts which were in scope of the performed audit Multiwrap, DropERC1155 and SignatureDrop.

Source Code

The following source code was reviewed during the audit:

Repository	Commit
Github (Multiwrap)	e33de553cfcdbcaa7c0a179756488b4e1238291a
Github (DropERC1155)	f10d5433f004260ed80ca877e5427fb273e2f40c
Github (SignatureDrop)	e1c2115c31a8be5e1453820b144c3ade01460f9a

Specifically, we audited the following contracts as part of Multiwrap contract audit:

Repository	Sha256
contracts/multiwrap/Multiwrap.sol	ceaaa52ceda0943f7fdf6044280189ca3a07bc8c8c5ee90d0aea3c29268f9a4b
contracts/feature/ContractMetadata.sol	df3db74a134e523735fc9915a8cd52f6d55dcad26fcbff4fd00e619f2a93bc7b
contracts/feature/Royalty.sol	f2ba6cef6221bc122452c8d7ba7aed1a70de6d52fcc9f280a85205c1440b3d79
contracts/feature/Ownable.sol	195496f2b9e8218a5e6bb92243ad9f6e5baa72104559807a38e069ca7c9257e5
contracts/feature/Permissions.sol	a2af3b9cdb65c69e3943113a824490c244e68a1e632c750a3b89d95f0c6186d6
contracts/feature/PermissionsEnumerable.sol	27e09155f457aa32cd1c51f892dbdee9806d7bfa9bc985b565283463a07b0dba
contracts/feature/TokenBundle.sol	492880c72765692ca59c1baecfa55d1a58753708a23377efbeff45793b055bc4
contracts/feature/TokenStore.sol	8b0ca57cbedbf8eb62b3ecd0a4e8bb51f845f26dabe70c41bd5056c9479d2517

contracts/lib/CurrencyTransferLib.sol	052c1c014b8169fdb02a9daa37b5edfbbbf9c883d89fcfe4ea3717810fecc76c
contracts/openzeppelin-presets/metatx/ERC2771ContextUpgradeable.sol	4ef0ce1601048c10a4b0fdc3247062be8f1a9ca0441c862ddfadc16251a31edb
contracts/interfaces/IMultiwrap.sol	d54f071277c95834259df0378bb569ce80132ba1adacb97a6eb71758395968b6
contracts/feature/interface/IContractMetadata.sol	453c5d2cecd21718181c667c95e89e0dc4e6ee0df3df7e2152f93ebdcbde06f2
contracts/feature/interface/IRoyalty.sol	6eb343aa794e6e30bbb1c8c7a6d09d8b380614dc6ca2ede1fb8d86908a38c409
contracts/feature/interface/IOwnable.sol	e588d8e1d498f6c1ea9cdc308914c8284a417cf3f18f9a2e9583111aa69962f0
contracts/feature/interface/IPermissions.sol	333d596baf00c08da55bc1671da3f5df65c4a1d9e8d5639e910d1c23ffb7f980
contracts/feature/interface/IPermissionsEnumerable.sol	5993fac74a2908a778d21786cf0542f32c8c57d05a03321175b630948bf4913e
contracts/feature/interface/ITokenBundle.sol	fe05e8c4123da579aab2a92efe43b925e81443c870ac05b0f3b99bcaee0321bb

We audited the following contracts as part of DropERC1155 contract audit

Repository	Sha256
contracts/drop/DropERC1155.sol	224b5233428ef803c6e875868945b840ec59f9694d1ce4dc42ee29b0e8fef582
contracts/lib/FeeType.sol	3d2ede585eb7e37872a0f3566a143f5b2aa586873160966d34c98963015f622d
contracts/lib/MerkleProof.sol	cf3d021220b40ba34a503595000419df6576fabb4309dc3c265abe4ad21a25c8
contracts/lib/CurrencyTransferLib.sol	052c1c014b8169fdb02a9daa37b5edfbbbf9c883d89fcfe4ea3717810fecc76c

contracts/openzeppelin-presets/metatx/ERC2771ContextUpgradeable.sol	4ef0ce1601048c10a4b0fdc3247062be8f1a9ca0441c862ddfadc16251a31edb
contracts/interfaces/IThirdwebContract.sol	8fc9d29ddee99b052ccdc521c272ee4df8a7de0e1754bfcba397dc5cdfa18c72
contracts/feature/interface/IPlatformFee.sol	a40ab9eb32bb694e01aed83c32e19e713f6686d5c10c41ceab2a962b65d954ae
contracts/feature/interface/IPrimarySale.sol	19fc349c2d09c7c3cf629010ac376f9e59876c753c7375dc0cd0d9962db2dea4
contracts/feature/interface/IRoyalty.sol	6eb343aa794e6e30bbb1c8c7a6d09d8b380614dc6ca2ede1fb8d86908a38c409
contracts/feature/interface/IOwnable.sol	e588d8e1d498f6c1ea9cdc308914c8284a417cf3f18f9a2e9583111aa69962f0
contracts/interfaces/ITWFee.sol	4c57ef2e5572551ee29ec7ecfcb67932f152f7b0ffd1e5c84e0976f577eb43c5
contracts/interfaces/drop/IDropClaimCondition.sol	acfcfa34578efe1c51d17c0506f3ee7261442bd6dcec49196a571918929c5a51
contracts/interfaces/drop/IDropERC1155.sol	440080243336aee49d674627c1a1dbc53fd7f75adc99bbebb93ee10f6a5d04c0

We audited the following contracts as part of SignatureDrop contract audit

Contract	Sha256
contracts/signature-drop/SignatureDrop.sol	b61014572ce0e07b44c5814570eb0efe23e9302c8660a5629f6cc47a3c983f6e
contracts/feature/ContractMetadata.sol	883965fe2c88a3ea36b56fbd780554485ee8c9bd5ac1d82f87dfa27cdf38820c
contracts/feature/PlatformFee.sol	5761f4a8b9a1bd90070a09091a94e50370616002ef0825299d54120324f7020d
contracts/feature/PrimarySale.sol	6f472f7d77830b4924862b9e33e1cea34a1d7be30cba0ca4d99b76acc63eee11
contracts/feature/Royalty.sol	3faf5a5fb83fafc6169f3d0a97d9186e5b3e

	0a178bbb99db3cb849691df3a87e
contracts/feature/DelayedReveal.sol	48df35ee1e617f6cd5ed52d1490719a12137ba77eb88df82aeed12140f3eceb8
contracts/feature/DropSinglePhase.sol	58af5a7c6e04de4cefb82f1d74a1f6c8875fc76469b05f3c595ca81faae1cae4
contracts/feature/LazyMint.sol	0f7aa682dd9c83e1b108d55c0a8b879dc4ee8fec582a9de3b36c3e24696d4d23
contracts/feature/Ownable.sol	fa86e93306669311a74343ad50cbe533442792f8091e810763dc6125fd710cb0
contracts/feature/Permissions.sol	e07a0b4d807e31b6297677887ad704e79e45cf15eecba710949d3a92d078ee69
contracts/feature/PermissionsEnumerable.sol	27e09155f457aa32cd1c51f892dbdee9806d7bfa9bc985b565283463a07b0dba
contracts/openzeppelin-presets/metatx/ERC2771ContextUpgradeable.sol	4ef0ce1601048c10a4b0fdc3247062be8f1a9ca0441c862ddfadc16251a31edb
contracts/lib/CurrencyTransferLib.sol	052c1c014b8169fdb02a9daa37b5edfbbbf9c883d89fcfe4ea3717810fecc76c
contracts/feature/SignatureMintERC721Upgradeable.sol	f83b0704e73d831f8d448a798c1a7eaf2b0dca156e276881c1cce925c3fd2c43
contracts/feature/interface/IClaimCondition.sol	0dbad456208d0d05608647c27de0aee95e92fd288e364cf552ecffe6aff2bcaa
contracts/feature/interface/IContractMetadata.sol	453c5d2cecd21718181c667c95e89e0dc4e6ee0df3df7e2152f93ebdcbde06f2
contracts/feature/interface/IDelayedReveal.sol	c6b5754ca0a19df8950b36b26ecef66b1c8408ed2dff305dbfbed9f4d9bf1e05
contracts/feature/interface/IOwnable.sol	e588d8e1d498f6c1ea9cdc308914c8284a417cf3f18f9a2e9583111aa69962f0
contracts/feature/interface/IPermissions.sol	333d596baf00c08da55bc1671da3f5df65c4a1d9e8d5639e910d1c23ffb7f980
contracts/feature/interface/IPermissionsEnumerable.sol	5993fac74a2908a778d21786cf0542f32c8c57d05a03321175b630948bf4913e
contracts/feature/interface/IPlatformFee.sol	a40ab9eb32bb694e01aed83c32e19e713f6686d5c10c41ceab2a962b65d954ae

contracts/feature/interface/IPrimarySale.sol	19fc349c2d09c7c3cf629010ac376f9e59876c753c7375dc0cd0d9962db2dea4
contracts/feature/interface/IRoyalty.sol	6eb343aa794e6e30bbb1c8c7a6d09d8b380614dc6ca2ede1fb8d86908a38c409
contracts/feature/interface/ISignatureMintERC721.sol	3fa03ed9c11deac6a8ab645465ee1b11604a7818cdb59b3ddc34c9b8dd5ec93e
contracts/feature/interface/IDropSinglePhase.sol	aa7a6dbeb9599756597bfc7426ed9331aaa6a8c977fb31b29defb721917dcc03
contracts/feature/interface/ILazyMint.sol	9cf7240f6527a848c1aa5267db2794fde9cbd8f11c3e5f9f6b0ac0ceca13eb4d

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.

Methodology

The audit was conducted in several steps.

First, we reviewed in detail all available documentation and specifications for the project, as described in the 'Specification' section above.

Second, we performed a thorough manual review of the code, checking that the code matched up with the specification, as well as the spirit of the contract (i.e. the intended behavior). During this manual review portion of the audit we primarily searched for security vulnerabilities, unwanted behavior vulnerabilities, and problems with systems of incentives.

Third, we performed the automated portion of the review consisting of measuring test coverage (while also assessing the quality of the test suite) and evaluating the results of various symbolic execution tools against the code.

Lastly, we performed a final line-by-line inspection of the code – including comments –in effort to find any minor issues with code quality, documentation, or best practices.

Issues Descriptions and Recommendations

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions. We highly recommend fixing the reported issue. If you have already deployed, you should upgrade or redeploy your contracts.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest. We recommend considering a fix for the reported issue.
Low	The risk is small, unlikely, or 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.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.
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.

[H-01] Wrapped ETH stuck in contract

~~HIGH~~ *Fixed by a39685a9a568ca19bf10ab98ff8b9c4fa6a3f311*

Multiwrap.sol supports receiving ETH by auto-wrapping incoming ETH to WETH. It does this by converting native tokens in CurrencyTransferLib through interaction with external WETH contract. After wrapping, the Multiwrap contract holds on to the wrapped native tokens until an unwrap is requested.

However, Multiwrap's WETH integration is missing a required `receive() external payable` function. When the user invokes `unwrap()`, for an asset with underlying ETH, it always reverts. Primary reason for that is the WETH contract cannot transfer back native tokens to Multiwrap due to missing `receive()`. As a result, the user's ETH is permanently stuck in the WETH contract, and the user cannot retrieve back his assets.

Consider implementing the `receive()` function to fix this issue.

[H-02] Batch reveal can be permanently corrupted

~~HIGH~~ *Fixed by 500f6562df3843cc1169dd983197071c0ab1adee*

In SignatureDrop.sol, the `reveal()` function is used to replace placeholder tokenBaseUri for a particular batch with final tokenBaseUri based on previously provided encrypted string. `reveal()` function is protected and callable by a user with privileged role `MINTER`. The `reveal()` function uses and relies on the `getRevealURI` function to retrieve decrypted final tokenBaseUri. For proper `reveal()`, `getRevealURI` must not revert.

However, in DelayedReveal.sol, `getRevealURI` is a public function and can be called by anyone. Also, this function can be successfully executed only once. The last line in this function modifies the state due to which all followup executions will revert. That would not be an issue if only legitimate invocation would be possible.

```
function getRevealURI(uint256 _batchId, bytes calldata
_key) public returns (string memory revealedURI) {
```

```
        bytes memory encryptedURI = encryptedBaseURI[_batchId];
        require(encryptedURI.length != 0, "nothing to
reveal.");

        revealedURI = string(encryptDecrypt(encryptedURI,
_key));

        delete encryptedBaseURI[_batchId];
    }
```

However, an attacker may simply invoke `getRevealURI` with **any key** to cause a permanently invalid contract state for a not yet revealed batch. That is because the `encryptDecrypt` function will return value even if an incorrect `_key` is provided by the caller.

Consider changing `getRevealURI` visibility to internal. In addition, consider introducing an extra argument to `getRevealURI`, e.g. `expectedRevealedURI` and corresponding guard condition to check if `expectedRevealedURI` matches `revealedURI` generated by `encryptDecrypt` method. This additional check may prevent contract owner from intentionally or accidentally breaking their batch reveal when they provide an incorrect decryption key.

[L-01] Public renounceRole() call can corrupt roleMembers state

~~LOW~~ *Fixed by e8d957936075f6fcc9b927a9c5b61c07b89db45b*

In Multiwrap.sol, an public invocation of `PermissionsEnumberable#renounceRole()` with a valid role argument can corrupt state in the `PermissionsEnumberable#roleMembers` variable for that particular role. Take the following example call trace:

```
PermissionsEnumerable#renounceRole(minter_role, Alice)
  Permissions#renounceRole(minter_role, account)
    Permissions#_revokeRole(minter_role, account)
```

```
PermissionsEnumerable#removeMember(minter_role, account)
```

And the following implementation of `removeMember()`:

```
function _removeMember(bytes32 role, address account) internal {
  uint256 idx = roleMembers[role].indexOf[account];
  delete roleMembers[role].members[idx];
  delete roleMembers[role].indexOf[account];
}
```

When `_removeMember()` is called with a valid role and unknown account, `idx` is 0, causing the contract to remove an unrelated member in the following line. This results in a corrupted state.

Consider updating `Permissions.sol#renounceRole` to check if the account actually has the role that is being renounced.

[L-02] Incorrect supportsInterface implementation

~~LOW~~ *Fixed by a3d7cc8403469061a89bdb82d742b6eb2adb4916*

In Multiwrap.sol, the `supportsInterface()` function overrides both ERC1155Receiver's and ERC721Upgradeable's implementations:

```
function supportsInterface(bytes4 interfaceId)
    public
    view
    virtual
    override(ERC1155Receiver, ERC721Upgradeable)
    returns (bool)
{
    return
        super.supportsInterface(interfaceId) ||
        interfaceId == type(IERC721Upgradeable).interfaceId ||
```

```
            interfaceId == type(IERC2981Upgradeable).interfaceId;
    }
```

Due to how multiple inheritance works in Solidity, calling `super` **will not** invoke the `supportsInterface()` implementations for *both* parent contracts. As a result, this contract **will not** be recognized as an ERC1155Receiver by external contracts, possibly blocking integration.

Consider updating `supportsInterface()` to properly advertise ERC1155Receiver support like so:

```
function supportsInterface(bytes4 interfaceId)
    public
    view
    virtual
    override(ERC1155Receiver, ERC721Upgradeable)
    returns (bool)
{
    return
        interfaceId == type(IERC2981Upgradeable).interfaceId ||
          ERC1155Receiver.supportsInterface(interfaceId) ||
          ERC721Upgradeable.supportsInterface(interfaceId);
}
```

[L-03] LazyMint of a new batch can affect previous batch

~~LOW~~ *Fixed by 382f23c6e1044e9d16dc847577bffbc3c75af81e*

In SignatureDrop.sol, the default contract admin can lazy mint a batch with 0 tokens by calling `lazyMint()` function. As a result, the internal identifier for the new empty batch becomes the same as the identifier for the previous batch. Due to this identifier overlap, followup actions targeting the new batch result in changes for the previous batch. This

allows an admin to overwrite `tokenBaseURI` for the previous batch maliciously or accidentally by calling `reveal()` for new batch as depicted in the following test:

```
function test_delayedReveal_withNewLazyMintedEmptyBatch() public {
    vm.startPrank(deployerSigner);

    bytes memory encryptedURI = sigdrop.encryptDecrypt("ipfs://",
"key");
    sigdrop.lazyMint(100, "", encryptedURI);
    sigdrop.reveal(0, "key");

    string memory uri = sigdrop.tokenURI(1);
    assertEq(uri, string(abi.encodePacked("ipfs://", "1")));

    bytes memory newEncryptedURI =
sigdrop.encryptDecrypt("ipfs://secret", "key");
    sigdrop.lazyMint(0, "", newEncryptedURI);
    sigdrop.reveal(1, "key");

    // token uri for token 1 is overwritten and it shouldn't
    string memory newUri = sigdrop.tokenURI(1);
    assertEq(newUri, string(abi.encodePacked("ipfs://secret", "1")));

    vm.stopPrank();
}
```

Consider adding a guard to prevent `SignatureDrop#lazyMint` being invoked with 0 `_amount`.

[L-04] Incorrect handling of invalid role approvals/removals

~~LOW~~ *Fixed by 0fb253fce0e728b3400c40f65e1e017a5807c22e*

Permissions.sol's implementation allows granting the same role to an account multiple times. Also, it allows removing a role from an account that doesn't have that role. This may result in unexpected `RoleGranted` and `RoleRevoked` event emissions. Moreover, it can introduce additional issues in child contracts, such as PermissionsEnumerable.sol, which are not expecting nor properly handling these cases.

Consider adding guards in Permissions.sol to prevent granting the same role to a particular account, and to prevent removing a role from an account that doesn't actually have the target role.

[L-05] Incorrect processing of role approval

~~LOW~~ *Fixed by c7ae40424b72eac1736184249cf45fd06ee1787e*

In SignatureDrop.sol, a call to `grantRole()` results in the `PermissionsEnumerable#_addMember()` internal function being called two times. As a result, the `roleMembers[role].members` storage variable contains unwanted duplicate records.

Consider updating `PermissionsEnumerable#grantRole` to not call `_addMember()`, since it will already be executed as part of downstream processing.

[L-06] claimCondition.startTimestamp is not enforced

~~LOW~~ *Fixed by e7a11f95e767c1deaa053a09d496984bac022568*

The SignatureDrop specification describes claimCondition.startTimestamp as follows:

> `The unix timestamp after which the claim condition applies. The same claim condition applies until the startTimestamp of the next claim condition.`

Based on the above description, SignatureDrop users may create a `claimCondition` to enable token claiming at a specific time in the future. However, in DropSinglePhase.sol's claim function, startTimestamp is not checked. This allows users to start claiming immediately, even if startTimestamp is set in the future.

Consider updating the implementation to check if startTimestamp condition has been satisfied or updating documentation related to startTimestamp to make it clear that it is not enforced.

[L-07] Unsafe usage of msg.value

~~LOW~~ *Fixed by ed6d60af9dd3c7acdb163416f5b5674e7db185f6*

Multiwrap.sol relies on `CurrencyTransferLib#transferCurrencyWithWrapper()` for proper operation. In this method, `msg.value` is used to check if necessary assets have been provided.

However, note that `transferCurrencyWithWrapper()` is called within a loop. Although not an issue today, if the parent contract later supports holding ETH via an upgrade, the new functionality may be vulnerable to having assets drained from the contract.

Consider not relying on `msg.value` directly in a library function which can be executed in a loop, and instead refactor code to execute necessary checks on a more higher/appropriate level.

[Q-01] Emitted TokensLazyMinted event does not match spec

~~CODE QUALITY~~ *Fixed by ac789394c99342f6e56497b14768a22e53061143*

In `SignatureDrop#lazyMint` method `TokensLazyMinted` event is emitted in following way

```
emit TokensLazyMinted(startId, startId + _amount, _baseURIForTokens,
_encryptedBaseURI);
```

DropERC721.sol another contract which has similar functionality emits this event in the following way. Notice difference in second argument.

```
emit TokensLazyMinted(startId, startId + _amount - 1,
_baseURIForTokens, _encryptedBaseURI);
```

Consider updating `TokensLazyMinted` event emission in `SignatureDrop#lazyMint` to match specification.

[Q-02] Upgradable contracts missing __gap variable

~~CODE QUALITY~~ *Acknowledged*

Upgradable contracts in the hierarchy of contracts need to have `__gap` variable in order for future changes not to break contract storage.

Response: Contracts aren't meant to be upgradeable and the missing `__gap` variable is intended.

[Q-03] Event indexing

~~CODE QUALITY~~ *Fixed by 076687de665b1d505ebbb2b2d777ed34b81d30bc*

Several events could benefit from indexing:

- `event OwnerUpdated - prevOwner` and `newOwner`
- `event TokensLazyMinted – startTokenId`
- `event TokenURIRevealed – index`
- `event DefaultRoyalty – newRoyaltyRecipient`
- `event RoyaltyForToken – royaltyRecipient`
- `event PlatformFeeInfoUpdated – platformFeeRecipient`
- `event TokensClaimed – startTokenId`

[Q-04] Natspec documentation

~~CODE QUALITY~~ *Fixed by 83c99cfe018bf2fe9a09731b3bb075e6327dbdd2*

Missing more detail natspec comments for some of the features (see `IClaimCondition.sol` as a reference):

- `IDelayedReveal.sol, DelayedReveal.sol`
- `IContractMetadata.sol, ContractMetadata.sol`
- `IDropSinglePhase.sol`
- `ILazyMint.sol, LazyMint.sol`
- `IOwnable.sol, Ownable.sol`
- `IPermissions.sol, Permissions.sol`
- `IPlatformFee.sol, PlatformFee.sol`
- `IPrimarySale.sol, PrimarySale.sol`
- `IRoyaltyInfo.sol, RoyaltyInfo.sol`

[Q-05] Change visibility from public to external

~~GODE QUALITY~~ *Fixed by ba4e4fe7054e0d5cc74c567ff37851429993d0ac*

Visibility for following methods can be changed from public to external:

- `Permissions#getRoleAdmin`
- `SignatureDrop#burn`

[G-01] Reduce the number of loops in Multiwrap#wrap and Multiwrap#unwrap

Status: Not fixing

Wrap executes three loops, all for iterating tokens.

- `1st loop - to check if asset is allowed`
- `2nd loop - wrap > _storeTokens > _setBundle()`
- `3rd loop - wrap > _transferTokenBatch`

All of the above can be combined in one loop, saving gas costs. The same can be said for unwrap as well, instead of 2 loops, there can be one.

Response: Not fixing, suggested optimization requires refactoring code across several levels of contract inheritance.

[G-02] Refactor TokenBundle#_setBundle()

Status: Fixed by e7a59e0089c568c9febb4b7b7ea22f2bc2ccaaf5

`TokenBundle#_setBundle` has a code path for updating the bundle, which is unused in Multiwrap's context. It's not only unused but it's also executed while creating a bundle. As a result, whenever this method is invoked an unnecessary condition is checked each time in the loop, increasing gas costs.

Consider creating two separate functions for create and update.

[G-03] Remove unnecessary checks in CurrencyTransferLib

Status: Fixed by fe70a1d5518c3e977270c3598caec1cfbb28bf42

Following optimizations are done in `CurrencyTransferLib`.

1. If `amount = 0 return`, in `transferCurrency` and `transferCurrencyWithWrapper`
2. If `sender = recipient` return, in `safeTransferERC20`

The optimizations done are logically correct. But the issue is that cases when these checks are satisfied are very rare, and optimizing for them, though saves gas costs for these edge cases, increases the gas costs for all other use cases.

Consider removing these optimizations.

[G-04] Reduce the length of string error messages

Status: Fixed by 913e513a70504436fd1385c8bc3e50c5a2ecb5ab

Reduce the length of string error messages to reduce contract size. Also consider using Solidity 0.8.4+ feature - Custom Errors .

[G-05] Return early in PermissionsEnumerable#getRoleMember

Status: Fixed by f3ad82c89af40edc0a86416b3c1a5d0b1773a05b

In method `PermissionsEnumerable#gerRoleMember`, return early when a match is found instead of iterating through the whole array on each invocation.

Automated Analysis

Slither

Slither is a solidity static analysis framework. It detects many vulnerabilities, from high threats to benign ones, of which there are usually many.

In order to run Slither against the codebase we ran the following command and filtered for relevant files:

- `$ slither .`

Slither identified many issues; manual inspection revealed that almost all of them to be false positives. However, [L-07] and [Q-05] have been confirmed as issues.

ERC721 conformance

In order to test ERC721 conformance of SignatureDrop.sol contract we ran the following command:

- `$ slither-check-erc --solc-remaps @=node_modules/@ --erc ERC721 contracts/signature-drop/SignatureDrop.sol SignatureDrop`

Resulting in following output:

```
## Check functions
[✓] balanceOf(address) is present
 [✓] balanceOf(address) -> (uint256) (correct return type)
 [✓] balanceOf(address) is view
[✓] ownerOf(uint256) is present
```

```
 [✓] ownerOf(uint256) -> (address) (correct return type)
 [✓] ownerOf(uint256) is view
[✓] safeTransferFrom(address,address,uint256,bytes) is present
 [✓] safeTransferFrom(address,address,uint256,bytes) -> () (correct
return type)
 [✓] Transfer(address,address,uint256) is emitted
[✓] safeTransferFrom(address,address,uint256) is present
 [✓] safeTransferFrom(address,address,uint256) -> () (correct return
type)
 [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
 [✓] transferFrom(address,address,uint256) -> () (correct return type)
 [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
 [✓] approve(address,uint256) -> () (correct return type)
 [✓] Approval(address,address,uint256) is emitted
[✓] setApprovalForAll(address,bool) is present
 [✓] setApprovalForAll(address,bool) -> () (correct return type)
 [✓] ApprovalForAll(address,address,bool) is emitted
[✓] getApproved(uint256) is present
 [✓] getApproved(uint256) -> (address) (correct return type)
 [✓] getApproved(uint256) is view
[✓] isApprovedForAll(address,address) is present
 [✓] isApprovedForAll(address,address) -> (bool) (correct return type)
 [✓] isApprovedForAll(address,address) is view
[✓] supportsInterface(bytes4) is present
 [✓] supportsInterface(bytes4) -> (bool) (correct return type)
 [✓] supportsInterface(bytes4) is view
[✓] name() is present
 [✓] name() -> (string) (correct return type)
 [✓] name() is view
[✓] symbol() is present
 [✓] symbol() -> (string) (correct return type)
[✓] tokenURI(uint256) is present
 [✓] tokenURI(uint256) -> (string) (correct return type)

## Check events
[✓] Transfer(address,address,uint256) is present
 [✓] parameter 0 is indexed
```

```
 [✓] parameter 1 is indexed
 [✓] parameter 2 is indexed
[✓] Approval(address,address,uint256) is present
 [✓] parameter 0 is indexed
 [✓] parameter 1 is indexed
 [✓] parameter 2 is indexed
[✓] ApprovalForAll(address,address,bool) is present
 [✓] parameter 0 is indexed
 [✓] parameter 1 is indexed
```

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.