macro

thirdweb A-12

Security Audit

June 19th, 2023
Version 1.0.0

Prepared for: thirdweb

Table of Contents

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

Introduction

This document details the results of the security audit for thirdweb's latest contract system, commonly referred to as the version 1 Base Contracts. The security audit was performed by the Macro security team from May 2 2023 to May 19, 2023.

The purpose of the 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 uncovered many issues and weaknesses, this document and Macro's subsequent recommendations cannot guarantee the security, or its single-sided 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 Fix	Addressed
High	1	-	-	1
Medium	3	-	-	3
Low	7	-	-	7
Code Quality	6	-	-	6
Gas Optimizations	1	-	-	1
Total	18	-	-	18

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 public handoff document provided through Notion.

Source Code

The following source code was reviewed during the audit:

- **Repository** [contracts](https://github.com/thirdweb-dev/contracts)

  Commit hash **(Smart Accounts)**: 3b5d5d5f7f7d9b0f2f1e4d8d9c3f9c1867d9f61

  Commit hash **(Open Edition ERC 721)**: 7d5a1d8f6a2d1f7c2d3e4b5a6f7c8d9e1f2a3b4c

We audited the following contracts specifically for the Smart Accounts audit:

Contract	SHA256
contracts/smart-wallet/utils/BaseAccountFactory.sol	2b2f5d2f1f6c9d3e8b4c5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c
contracts/smart-wallet/utils/BaseAccount.sol	17a1cf7e9d8b2c3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e
contracts/smart-wallet/utils/TokenCallbackHandler.sol	9f8e7d6c5b4a3f2e1d0c9b8a7f6e5d4c3b2a1f0e9d8c7b6a5f4e3d2c1b0a9f8e
contracts/smart-wallet/utils/Helpers.sol	1f7f7f9a3b8e4d2c6f5a9b7e3d1c8f4a6b2e9d5c7f3a8b6e4d2c9f1a5b7e3d8c
contracts/smart-wallet/utils/EntryPointToken/EntryPointToken.sol	8f6f1c9d4e7a2b5f3c8e6d9a4b7f2c5e8d1a6b3f9c4e7d2a5b8f6c1e9d4a7b3f
contracts/smart-wallet/utils/EntryPointToken/IEntryPointToken.sol	3b9c1d6e4f7a2b5c8d3e6f9a4b7c2e5d8f1a6b3c9e4d7f2a5b8c6e1d9f4a7b3c
contracts/smart-wallet/utils/TokenPaymaster/TokenPaymasterErrors.sol	4d7a9f2c6e1b8d5a3f7c9e4b6d2a8f5c1e7b3d9a4f6c2e8b5d1a7f3c9e6b4d2a
contracts/smart-wallet/utils/TokenPaymaster/TokenPaymasterStorage.sol	6e2b8d4f1a7c9e3b5d6a2f8c4e1b7d9a3f5c6e2b8d4a1f7c9e3b5d6a2f8c4e1b
contracts/smart-wallet/utils/TokenPaymaster/TokenPaymaster.sol	2a5f8c1d9e4b7a3f6c2e8d5b1a9f4c7e3b6d2a8f5c1e9b4d7a3f6c2e8d5b1a9f
contracts/smart-wallet/utils/BasePaymaster.sol	9d3e7b1a5f8c2e6d4b9a3f7c1e5d8b2a6f4c9e3d7b1a5f8c2e6d4b9a3f7c1e5d
contracts/smart-wallet/utils/BaseAccountFactory.sol	5c8a2f6e1d9b4a7c3f5e8d2b6a1f9c4e7b3d5a8f2c6e1d9b4a7c3f5e8d2b6a1f
contracts/smart-wallet/utils/BaseAccount.sol	7f1b4e9d2a6c8f3b5e7d1a9c4f2b6e8d3a5f7c1b4e9d2a6c8f3b5e7d1a9c4f2b
contracts/smart-wallet/utils/AccountPaymaster/AccountPaymasterStorage.sol	8e4d6a2f9c1b7e3d5a8f4c6e2b9d1a7f3c5e8b4d6a2f9c1b7e3d5a8f4c6e2b9d
contracts/smart-wallet/utils/AccountPaymaster/AccountPaymaster.sol	1c7f3a9e5b2d6f8c4a1e7b3d9f5c2a6e8b4d1f7c3a9e5b2d6f8c4a1e7b3d9f5c

We audited the following contracts specifically for the Open Edition ERC 721 audit:

Contract	SHA256
contracts/open-edition/OpenEdition.sol	5c2b1c7d3a9f8e6b4d2a1f7c9e3b5d8a6f4c2e9b7d3a1f5c8e6b4d2a9f7c3e1b

Note: This document contains an audit summary 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

Listed are issues/suggestions in a scroll down format as follows:

🟥 Any signer can withdraw the smart wallet's deposit amount

🟧 Any signer can corrupt the AccountFactory's signers registry

🟨 Initial permissions are not enumerable

🟨 Nonce tokens can get locked in NonceBasedLocalFactory

🟨 Potential for storage collision in NonceBasedLocal

🟨 Nonce tokens can get locked in OpenEditionERC721

🟩 Redundant gasfee Recovery during execution

🟩 Incorrect supportsInterface implementation

🟩 Potentially exploitable if signature malleability

🟩 Unused imports

🟩 Unused Errors

🟩 Misleading variable naming

🟩 Unused Import

🟩 Executor contract

🟩 Use optimized Clones version

Security Level Reference

We quantify issues in three parts:

1. The high-medium-low-none rating **Impact** of the issue.
   - Low 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 bad or good thing to happen
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
0 - Critical Severity	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant functionality fails to be met.** We recommend the client **must** address the issue, no matter what, because not fixing would let key bad, or some bad-looking things happen, or the project's reputation is against the provided spec.
1H - High Severity	We recommend the client **must** address the issue, no matter what, because not fixing would let key bad, or some bad-looking things happen, or the project's reputation is against the provided spec.
1M - Medium Severity	We recommend the client **is seriously consider** fixing the issue as the implications of not fixing the issue are severe enough to impact the project significantly, albeit not in a catastrophic way.
1L - Low Severity	The fix is small, unlikely, or may not relevant to the project in the foreseeable future. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
2G Q - Code Quality	The issue identified does not pose any obvious risk, but may could improve overall code quality, on-chain composability, future upgradeability, or plain-vanilla speed of product delivery.
2G I - Informational	Warnings and things to keep in mind when operating the protocol, No immediate action required.
2G G - Gas Optimizations	The presented optimizations suggested would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

🟥 **Any signer can withdraw the smart wallet's deposit amount**

Location	Impact	Likelihood	Severity
Gnosisfriendly Design	Critical	High	Medium

Description

The spec states the following for the `DEFAULT_ADMIN_ROLE`:

> Only a holder of this role can withdraw the smart wallet's gas deposit from the EntryPoint contract.

Unless it is possible for some signer to withdraw the smart wallet's deposit amount from the `EntryPoint`, contrary to the outstanding a user operation that calls `withdrawGasReserve()` directly from the smart wallet.

The Nonces spec states to all three wallet versions: `BasicAccount`, `NonceBasedLocal`, and `NonceSessionLocal`.

Remediations to Consider

Consider adding logic to the `validate` and `executeUserOp` function that prevents signer from calling `withdrawGasReserve()` directly from the smart wallet. Consider implementing certain function selectors that are not allowed to be called via `execute` and `executeBatch`. Functions.

🟧 **Any signer can corrupt the AccountFactory's signers registry**

Location	Impact	Likelihood	Severity
Gnosisfriendly Design	Critical	High	Medium

Description

`NonceSessionFactory` implements `addSigner` and `removeSigner` functions to track the registered signers of an account. However, any signer can add and remove signers as they wish by submitting a user operation that calls `NonceSessionFactory.addSigner()` or `NonceSessionFactory.removeSigner()` directly from the smart wallet. This can lead to various issues such as:

- Admin can grant a new signer the `SIGNER_ROLE`, as the signer has been already granted the signer signer in the factory's signers registry.
- `NonceSessionFactory` wrongly reports registered signers in `getSignersOfAccount()`. Off-chain components and other protocols may rely on the information which can lead to various issues when returning data is not accurate.

The behavior applies to all three wallet versions: `BasicAccount`, `NonceBasedLocal`, and `NonceSessionLocal`.

Remediations to Consider

Consider adding logic added to the suggested remediation in 01-01 so that `addSigner()` and `removeSigner` functions can be called via a user operation transaction.

🟨 **Initial permissions are not enumerable**

Location	Impact	Likelihood	Severity
Core Cases	Medium	High	Medium

Description

`AccountFactory._createAccount` is called within the `initialize` functions of Dynamicaccount and `NonceBasedLocalFactory.createAccount`. `AccountFactory.initialize` function is shaping like:

```
function initialize(bytes[] data, address[] ,address[] ,address[] ,address[],
bytes, ,bytes[] ,bytes[] ,bytes[]) {
}
```

In contrast, the `AccountFactory` and inherits from `MultiCallUpgradeable` using `__gnosisSafeMultiCallUpgradeable_init`. As a result, roles added in the initialize function of `DynamicAccount` and `NonceBasedLocal` will not show up in the respective `RoleGrantedEvent`s. Functions like `getRoleMember` and `getRoleMemberCount`.

Remediations to Consider

Use `MultiCallUpgradeable` storage location in `AccountFactory._createAccount` function.

🟨 **Native tokens can get locked in NonceBasedLocalFactory**

Location	Impact	Likelihood	Severity
Core Cases	Medium	Low	Medium

Description

`NonceBasedLocalFactory` inherits from `BaseAccount` which implements the `fallback()` and `receive()`. functions allowing the contract to receive native tokens. This allows to send native tokens to the factory address. However, the factory does not have functionality to transfer out those tokens, leading trapped tokens in the contract forever.

Remediations to Consider

Consider overriding `fallback()` and `receive()` functions in `NonceBasedLocalFactory`, so that the transaction reverts when native tokens are accidentally transferred to the contract.

🟨 **Potential for storage collision in NonceBasedLocal**

Location	Impact	Likelihood	Severity
Upgradeability	Medium	Low	Medium

Description

In NonceBasedLocal, the state variable `factory` doesn't use the "storage-slot" pattern.

This causes the potential for storage collisions to occur, since the contract's storage slots pattern and use delegate calls to read and write to storage. Storage collisions can allow for attacks to occur, such from bad-faith to fully-trustless coupling between upgrade and vulnerabilities.

Note that for a storage collision to occur, it is required that an extension is added that also doesn't use the "storage-slot" pattern.

Remediations to Consider

Consider using the "storage-slot" pattern for setting the `factory` variable.

🟨 **Native tokens can get locked in OpenEditionERC721**

Location	Impact	Likelihood	Severity
Core Cases	Medium	Low	Medium

Description

`OpenEditionERC721` implements the `payMint()` function (no event) contract that accepts native tokens. `payMintTokenAddress` is intended parameter for `PaymentSplitter.releaseToken()`.

The issue of stuck-by native tokens in the contract occurs under the following circumstances:

1. Payment currency is `ClaimCondition[0]`, is set to ERC20.
2. A user calls `payMint` with appropriate amount of ERC20 tokens approved and accidentally also passes native tokens along.

As a result, the `payMint` call succeeds but the native tokens passed along will be locked within the `OpenEditionERC721` contract.

Note that this issue also applies to other Thirdweb contracts that inherit from the `Drop` extension, specifically the `_collectPriceOnClaim` function. `_collectPriceOnClaim` `_transferSupportedTokens`.

Remediations to Consider

Consider checking that its native tokens are transferred if `msg.value` is 0 when payment currency is set to ERC20.

🟩 **BaseRouter's getter functions don't include default extension**

Location	Impact	Likelihood	Severity
Interoperability	Low	High	Low

Description

The BaseRouter implementation used by Dynamicaccount and ManagedAccount doesn't consider the default extensions in its getter functions:

- getAllFunctionsOnChain()
- getFunctionSet()
- getFunctionsInExtensionOnChain()
- getAllFunctionsOfSignature()

This differs from the behavior of dynamic contracts that use the BaseRouter implementation, which does include default extension. Consider adding extensions will be returned in all BaseRouter's getter functions.

Remediations to Consider

Consider including the default extension in the BaseRouter's getter functions to have consistent behavior among different implementations of BaseRouter.

🟩 **Incorrect supportsInterface Implementation**

Location	Impact	Likelihood	Severity
Interoperability	Medium	Low	Low

Description

In BaseRouter.sol, the `supportsInterface` function is implemented as follows:

```
function supportsInterface(bytes interfaceId) public view virtual returns (bo
ool[] interfaceId) {
    return supportsInterfaceStorage(getRouterStorage().data, interfaceId);
}
```

Whereas the inherited extension AccountExtension.sol implements its own `supportsInterface` function to signal the `IERC165InterfaceId` and `IERC165InterfaceId` as implemented:

```
function supportsInterface(bytes4 interfaceId) public view virtual override(ER
C165,IERC165) {
    return
        interfaceId == type(IERC165).interfaceId ||
        interfaceId == type(IERC165Upgradeable).interfaceId ||
        super.supportsInterface(interfaceId);
}
```

However, the `supportsInterface` implementation in `AccountExtension` will never be hit as it is already implemented in `BaseRouter`. As a consequence, when `supportsInterface` is called for ManagedAccount and DynamicAccount, it wrongly returns `false` when called with `IERC1155Receiver` and `IERC721Receiver` interface ids.

Remediations to Consider

Consider changing `BaseRouter.supportsInterface` to return true for `IERC1155Receiver` and `IERC721Receiver` interface ids.

Q-1 Potentially susceptible to signature malleability

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed	Low

A vulnerable version of ECDSA.sol is used that is susceptible to signature malleability. See details here. The current version of ECDSA.sol being used is identical to Openzeppelin's version 7.0 whereas the vulnerability has been patched in 7.3.

Contracts that are affected by this vulnerability are those that implement replay protection by marking the signature itself as used instead of any nonce or unique id.

Throughout the Thirdweb code base - whenever ECDSA.sol is used - replay protection happens either via checking the nonce or a unique id. Thus, with current usage of ECDSA.sol, there is no immediate security risk imposed.

Consider updating ECDSA.sol to reflect changes in patched Openzeppelin's version 7.3. This ensures that no vulnerabilities will be introduced by using ECDSA.sol, regardless of the replay protection being used.

Q-2 Unused imports

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed	Low

The following imports are not required and can be removed:

- In DynamicAccount.sol:

```
import "../non-upgradeable/Account.sol";
```

- in DynamicAccountFactory.sol:

```
import "../utils/BaseAccount.sol";
import "../../openzeppelin-presets/proxy/Clones.sol";
```

- in ManagedAccountFactory.sol:

```
import "../utils/BaseAccount.sol";
import "../../openzeppelin-presets/proxy/Clones.sol";
```

Q-3 Cleanup import

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed	Low

EntryPoint.sol contains the following import:

```
import ".//Exec.sol";
```

Consider removing one `/` from the above import line.

Q-4 Misleading variable names

TOPIC	STATUS	QUALITY IMPACT
Naming	Fixed	Low

`BaseAccountFactory.addSigner` contains the following lines:

```
bool isAlreadyAccount = accountsOfSigner[_signer].add(account);
bool isAlreadySigner = signersOfAccount[account].add(_signer);
```

The variable names `isAlreadyAccount` and `isAlreadySigner` are misleading, as `EnumerableSet.add` function returns `true` when a new address is added; and `false` when the address is already in the mapping.

Consider renaming above variables to something more appropriate such as:

- `isNewAccount`
- `isNewSigner`

Q-5 Unused import

TOPIC	STATUS	QUALITY IMPACT
Best Practices	Fixed	Low

The following import is not required and can be removed:

- In OpenEditionERC721.sol:

```
import "./extension/DelayedReveal.sol"
```

Q-6 Inaccurate comment

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	High

The following comments in `OpenEditionERC721` are incorrect:

1. For `minterRole` variable declaration:

```
/// @dev Only MINTER_ROLE holders can sign off on `MintRequest`s and lazy mint
```

1. For `_canSetSharedMetadata` function:

```
/// @dev Returns whether lazy minting can be done in the given execution contex
```

Lazy mints are mentioned on both of the above comments which are not supported in `OpenEditionERC721`. Consider changing the comments appropriately.

G-1 Use optimized `Clones` version

TOPIC	STATUS	GAS SAVINGS
Best Practices	Fixed	Medium

The OpenZeppelin's Clones.sol contract has been updated to save gas on clone creation (see here). The version referenced by Thirdweb is still using the unoptimized version.

Consider updating Clones.sol to latest version to save gas costs.

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.