OpenZeppelin | security

OpenZeppelin Community Contracts Audit

April 3, 2025

Table of Contents

Summary

Type	Library
Timeline	From 2025-03-13 To 2025-03-21
Languages	Solidity

Total Issues	15 (12 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	4 (3 resolved)
Notes & Additional Information	8 (6 resolved)

Scope

We audited the OpenZeppelin/openzeppelin-community-contracts repository at commit 18db32c.

In scope were the following files:

```
contracts
├── account
│   ├── Account.sol
│   ├── AccountCore.sol
│   └── extensions
│       ├── AccountERC7579.sol
│       ├── AccountERC7579Hooked.sol
│       └── ERC7821.sol
├── token/ERC20/extensions/ERC20Bridgeable.sol
└── utils/cryptography
    ├── AbstractSigner.sol
    ├── ERC7739.sol
    ├── ERC7739Utils.sol
    ├── SignerECDSA.sol
    ├── SignerERC7702.sol
    ├── SignerP256.sol
    └── SignerRSA.sol
```

Update: *All resolutions and the final state of the audited codebase mentioned in this report are contained at commit 630ba44. In addition, at the commit 139b11f, the* `Account.sol` *contract was removed and* `AccountCore.sol` *was renamed to* `Account.sol`*. Files that imported these files were updated accordingly.*

Please note that changes introduced to contracts not in-scope for this audit were not part of the additional reviews.

System Overview

Accounts

The `AccountCore.sol` abstract contract is a simple account implementation that is ERC-4337 compliant. This contract is meant to be unopinionated and has no mechanism for performing external calls. Implementing this mechanism is left to the developer inheriting from this contract, who also is able to choose the desired signing mechanism. The `Account.sol` is an abstract contract meant to be a more opinionated account implementation that also inherits from ERC-7739 to prevent signature replayability between smart accounts.

As its name suggests, the `AccountERC7579.sol` abstract contract complies with ERC-7579 and offers developers a modular option for smart accounts. This contract, by default, forces the validation to come from the validation module rather than a signer. The `AccountERC7579Hooked.sol` abstract contract extends the `AccountERC7579.sol` contract to add hook modules on top the validation, execution, and fallback ones. Lastly, the `ERC7821.sol` abstract contract adds functionality for batch execution following the ERC-7821 specifications.

`ERC20Bridgeable`

The `ERC20Bridgeable.sol` abstract contract implements the ERC-7802 token interface for better cross-chain interoperability. This bridgeable ERC-20 token contract is meant to be deployed on several different chains that may come with different security underpinnings. The implementation itself has no distinction between the native chain, i.e. the chain where the token originated from, and non-native chains, i.e. the chains where the token can be bridged to.

When inheriting this contract, consider these two cases separately to avoid inconsistencies from independently running token contracts on several chains. For instance, if the total supply of a token needs to be capped, one should avoid (pre)minting or burning on non-native chains for consistent accounting on the native chain.

ERC7739

The `ERC7739.sol` abstract contract and the `ERC7739Utils` library allow developers to easily build a smart account that follows ERC-7739. The motivation behind ERC-7739 is to prevent signature replayability by including information such as the address and chain ID of the account itself in an EIP-712-compliant manner.

Signers

The `AbstractSigner` is an abstract contract that all signers implement. It has one function, `_rawSignatureValidation`, that must be implemented by the child contracts. The other signer contracts provide different signing methods, including ECDSA, P256, and RSA, and use OpenZeppelin cryptography libraries accordingly to perform the verification.

Security Model and Trust Assumptions

The ERC-4337 accounts depend on the `Entrypoint` singleton contract to function correctly and for the address to be set correctly, as there are protected functions that are protected by the `onlyEntryPoint` and `onlyEntryPointOrSelf` modifiers. The ERC-7579 accounts have a similar trust assumption as only the entry point or the account contract itself can install and uninstall the different modules. Furthermore, the modules themselves need to be trusted by the account installing them so as to not be malicious and to follow the appropriate specifications.

The `ERC20Bridgeable.sol` contract allows trusted token bridge(s) to mint and burn any user's balance on this token contract. Hence, the bridging mechanism, such as token address matching, L1 contract address aliasing, failed token recovery, bridging initiation, and withdrawal proofs, etc. on each deployed chain should be examined before giving the bridge(s) powerful minting/burning privileges.

The trusted bridge address may be updated over time. Thus, the `ERC20Bridgeable.sol` contract can be expected to be used in combination with certain access control mechanisms to ensure that the most up-to-date and correct bridge address is used for each interfacing chain at all times.

In addition, care should be taken when building custom integrations using other templates like `ERC20Paused` or `ERC20Capped` since such features rely on invariants that might not hold anymore or might interfere with bridging. The only privileged party is the bridge address, being in charge of calling the `crossChainMint` and `crossChainBurn` functions.

The signer contracts depend on a state variable that is set using the `_setSigner` function. This variable is a public key that is stored inside the contract. It is advised to call the `_setSigner` function during construction or initialization to prevent front-running.

High Severity

H-01 `ERC7821` Does Not Replace `address(0)` With `address(this)`

ERC-7821 proposes a standard to be used with EIP-7702 for EOAs to perform atomic batched execution. The specifications of ERC-7821 state that "For calldata compression efficiency, if a `Call.target` is `address(0)`, it will be replaced with `address(this)`". However, this replacement is not done. Worse, calls to `address(0)` are made as low-level calls and would thus silently fail while burning `msg.value`.

The original intention behind the EIP was for users to be able to save gas by using calls to `address(0)` as a way for the delegated account to recursively call itself. As the `ERC7821` contract is `abstract`, the exact impact would depend on the implementation used and whether such recursive calls are to be expected. Usage of this feature may result in loss of funds, or having transactions silently failing with undefined consequences.

Consider replacing `address(0)` with `address(this)` when executing a batch of calls to better match the specifications and prevent any loss of funds.

Update: *Resolved in pull request #5602 at commit e8fbea2 and in pull request #100 at commit 0d692f5. The Solidity Contracts team stated:*

> *Fixed in OpenZeppelin Contracts along with a dependency update to OpenZeppelin Community Contracts where the ERC-7821 implementation is hosted.*

Medium Severity

M-01 Modules Can Be Uninstalled With the Wrong `moduleTypeId`

When uninstalling a module, there is no validation that the `moduleTypeId` is supported. Passing an unsupported type ID (e.g., `MODULE_TYPE_HOOK` for a non-hooked `AccountERC7579`) makes it possible to call `onUninstall` on any address.

This could notably result in calling `onUninstall` on an installed module without the installed module being correctly removed from storage, which would put the account in an incorrect state. This would also incorrectly emit the `ModuleUninstalled` event, which could mislead trackers of such events. While unlikely, users mistakenly uninstalling a module with the wrong module ID may end up bricking their account.

Consider validating that the module type is supported when uninstalling it.

Update: *Resolved in pull request #99 at commit 8e3aec7.*

M-02 `isValidSignature` Forwards `address(this)` Instead of `msg.sender`

The `isValidSignature` function adds support for EIP-1271 signature validation following the EIP-7579 specifications. The `signature` argument is decoded and the inner signature is forwarded to a validator module.

The specifications state that "If ERC-1271 forwarding is implemented, the validator MUST be called with `isValidSignatureWithSender(address sender, bytes32 hash, bytes signature)`, where the `sender` is the `msg.sender` of the call to the smart account". This is also what is done in the standard ERC-7579 implementation. However, the `sender` is forwarded as being `address(this)` instead of `msg.sender`, which could cause unexpected results when validating signatures.

Consider either documenting this choice to move away from the specifications, or correcting the implementation to forward `msg.sender`.

Update: *Resolved in pull request #99 at commit d5a8c41.*

Low Severity

L-01 L1 Contract Address Aliasing On L2

On L2 rollup chains, it is a common practice to have address aliasing for L1 contracts, see for example Optimism. Hence, when the `ERC20Bridgeable.sol` contract is deployed on an L2

chain, one needs to be aware that the `msg.sender` parameter can be an aliased L1 contract address, an L2 contract address, or an L1/L2 EOA address.

While this does not directly pose a threat to the `ERC20Bridgeable.sol` contract, usage of plain `msg.sender` instead of the standard `Context.sol` pattern of using a re-implementable internal `_msgSender` function limits the flexibility of potentially unaliasing the caller's address if coming directly from an L1. Developers must be aware of this, especially when building custom integrations that might happen to have `msg.sender` being an aliased address.

Consider either using `_msgSender()`, leaving a note for developers to handle internal logic for unaliasing if needed. Alternatively, consider documenting the usage of `msg.sender` against aliasing edge cases.

Update: *Resolved in pull request #99 at commit 990d737.*

L-02 Missing and Misleading Documentation

Throughout the codebase, multiple instances of missing and/or misleading documentation were identified:

- In line 59 and in line 62 of `ERC7739Utils.sol`, the comment calls the separator the `DOMAIN_SEPARATOR`, but in the ERC-7739 spec, it is named `APP_DOMAIN_SEPARATOR`. Consider updating the comment to call it `APP_DOMAIN_SEPARATOR` for improved clarity.
- In line 61 of `ERC7739Utils.sol`, the comment states that this is the "original signature for the nested struct hash that includes the contents hash", but this would be more clear if it is explicitly stated that this is the signature over the `finalTypedDataSignHash`.
- In line 62 of `ERC7739Utils.sol`, the comment states that this is the separator of the "smart contract verifying the signature", but it would be clearer to say the "application smart contract" instead.
- In line 126 of `ERC7739Utils.sol`, the comment says "string-string-bytes32-string-bytes", but it should say "string-string-bytes32-bytes" instead to reflect the inputs of the `typedDataSignStructHash` function.
- In line 143 and in line 169 of `ERC7739Utils.sol`, the variable is named `contentsTypeName`, but the name could be changed to `contentsName` to align with the ERC-7739 spec.

- The `decodeTypedDataSig` function in `ERC7739Utils.sol` could use documentation which states that upon a failure to decode, it returns empty data instead of reverting.
- In line 16 of `Account.sol`, the comment suggests that it implements ERC-7821, but it does not. Consider removing this comment or clarifying that it would need to be inherited.
- In line 41 of `ERC7739.sol`, it is stated that "we return the magic value too", but this could be changed to "we return the magic value of `0x77390001`" since there are two potential magic values here, the aforementioned one and the magic value from ERC-1271.
- In lines 19-20 of `AccountERC7579.sol`, the comment mentions that this contract implements ERC-7739, but it does not. Consider removing this comment.
- In lines 170-171 of `AccountERC7579.sol`, the warning regarding combining with ERC-7739 could also be displayed in the NatSpec of the contract at the top of the file for more visibility.
- In line 227 of `AccountERC7579.sol`, the description of the `bytes memory initData` as being a "tuple" is unclear, as it could be interpreted to mean the abi encoding of a tuple. The formulation could be changed to "the `initData` is expected to be the concatenation of a 4-byte selector and the rest of the data".
- In lines 335-351 of `AccountERC7579.sol`, the comment that gives examples of similar behavior in other codebases could also list the example of the ERC-7579's reference implementation [1] [2].
- In `AccountERC7579.sol`, consider adding documentation for its lack of support for staticcall callType `0xfe`
- In `AccountERC7579.sol`, the documentation could add a warning that users should be careful if uninstalling the last validator module as it could brick the account.
- In lines 7-9 of `ERC7821.sol`, the comment states that this contract "only supports basic mode (no optional "opData")", but could clarify that this "only supports single batch with mode 0x01000000000000000000 (no optional "opData")".
- In lines 27-28 of `SignerECDSA.sol`, lines 27-28 of `SignerP256.sol`, and lines 27-28 of `SignerRSA.sol`, the comment states the phrase "avoiding to call", but this could be changed to "not calling" to make the point clear that the `_setSigner` function should indeed be called during construction or initialization to prevent front-running.
- The link format used to link to some EIPs leads to 404 errors. For example, "https://eips.ethereum.org/EIPS/eip-7766[ERC-7766]" linked in the `draft-IERC4337` contract.
- In the `_rawSignatureValidation` function in `SignerRSA.sol`, the hash that is passed as input into RSA represents the message that was signed to produce the given signature that should be verified. This hash can be the output of any hash function (e.g.,

Keccak, SHA256, etc). Through the subsequent call to `pkcs1Sha256`, this hash is hashed again with SHA256. While hashing the already hashed message seems counterintuitive, this process simply follows the steps of the RSASSA-PKCS1-V1_5-VERIFY procedure (specifically, Step 3: EMSA-PKCS1-v1_5 encoding, where the message is hashed in Step 1.), keeping in mind that the input hash is to be treated as the message that was signed. Consider adding documentation for this behavior in `_rawSignatureValidation` for improved clarity.

Consider adding the missing comments and revising the aforementioned ones to improve consistency and more accurately reflect the implemented logic, making it easier for auditors and other parties examining the code to understand what each section of code is designed to do.

Update: *Resolved in pull request #99 at commit c978c1f and at commit cc6e615. The Solidity Contracts team stated:*

> - *Account.sol was removed (and AccountCore.sol took its place)*
> - *Link syntax corresponds to the doc generation system. It is not markdown format.*

L-03 Type Name May Start Lowercase

The `decodeContentsDescr` function of `ERC7739Utils` retrieves the `contentsTypeName` and `contentsType` from the provided `contentsDescr`. The ERC-7739 spec has the following recommendation towards this content type name:

> For safety, it is RECOMMENDED to treat the signature as invalid if any of the following is true:
>
> - `contentsName` is the empty string `(i.e. bytes(contentsName).length == 0)`.
> - `contentsName` starts with any of the following bytes `abcdefghijklmnopqrstuvwxyz(`.
> - `contentsName` contains any of the following bytes `, )\x00`.

While the first and third conditions are checked, the function lacks a check that `contentsTypeName` does not start with a lowercase character.

Consider making sure that `contentsTypeName` starts with an upper-case character to enforce the recommendation of the spec. Furthermore, consider updating the comment on lines 161-162 to reflect this recommendation.

Update: *Acknowledged, not resolved. The Solidity Contracts team stated:*

> *The second recommendation is one that we discussed (at length) with the other ERC-7739 authors, and that we disagrea with. There is nothing in EIP-712 that prevents the contentsName to start with a lowercase letter. If an application uses such a contentsName, and request a signature, implementing the second recommendation would prevent such a signature from being produced. This would prevent the account from interracting with the application, possibly resulting in loss of funds (or similar). Therefore, we decided to purposefully not implement this recommendation.*

L-04 Specification Mismatch in `isValidSignature` Method

The `isValidSignature` function returns either the ERC-1271's magic value or `bytes4(0xffffffff)`.

```
function isValidSignature(bytes32 hash, bytes calldata signature) public view virtual
override returns (bytes4) {
    // check signature length is enough for extraction
    if (signature.length >= 20) {
        (address module, bytes calldata innerSignature) =
_extractSignatureValidator(signature);
        // if module is not installed, skip
        if (isModuleInstalled(MODULE_TYPE_VALIDATOR, module, Calldata.emptyBytes())) {
            // try validation, skip any revert
            try IERC7579Validator(module).isValidSignatureWithSender(address(this),
hash, innerSignature) returns (
                bytes4 magic
            ) {
                if (magic == IERC1271.isValidSignature.selector) return magic;
            } catch {}
        }
    }
    return bytes4(0xffffffff);
}
```

However, the specifications state that "The smart account's ERC-1271 `isValidSignature` function SHOULD return the return value of the validator that the request was forwarded to." While unlikely, it is possible that the validator may return something other than the ERC-1271 magic value or `bytes4(0xffffffff)`, in which case it would not be forwarded.

Consider returning the return value of the validator to better match the specifications.

Update: *Resolved in pull request #99 at commit 292d805.*

Notes & Additional Information

N-01 Missing IERC20 Interface Support

The supportsInterface function returns `true` for the IERC7802 and IERC165 interfaces but not for IERC20. In the case where the token bridge checks for IERC20 interface support, this may result in an unsuccessful bridging operation.

Consider returning `true` for `type(IERC20).interfaceId` as well.

Update: *Acknowledged, not resolved. The Solidity Contracts team stated:*

> *IERC20 interfaceId is not something common/standard. ERC-20 doesn't require this to be exposed, and a vast majority of ERC-20 contracts don't expose it (they don't include ERC-165 at all). The consequence is that since it is not exposed, it cannot reliably be checked/looked-up. For this reason, we prefer to not add the cost of exposing it knowing it will most likely never be checked. Note that the ERC document (that includes a reference implementation) does not mention exposing it https://eips.ethereum.org/EIPS/eip-7802.*

N-02 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In line 107 of `ERC7739Utils.sol`, "return" should be "returns".
- In line 188 of `ERC7739Utils.sol`, "read contentsTypeName for the end" should be "read contentsTypeName from the end".
- In line 98 of `AccountCore.sol`, "implementation" should be "implementations".
- In line 12 of `Account.sol`, "feature" should be "features" and "implementation" should be "implementations".
- In line 14 of `Account.sol`, "ERC-712" should be "ERC-721" while the word "transfers" is used twice.
- In line 22 of `AccountERC7579.sol`, "these functionality" should be "this functionality".
- In line 343 and line 366 of `AccountERC7579.sol`, "replicated" should be "replicates".
- In line 382 of `AccountERC7579.sol`, "we would could use" should be "we could use".

- In line 21 of `ERC7821.sol`, "should be implement by" should be "should be implemented by".
- In line 12 of `SignerECDSA.sol`, line 12 of `SignerP256.sol`, and line 12 of `SignerRSA.sol`, "an" should be "a".
- In line 13 of `SignerECDSA.sol`, line 13 of `SignerP256.sol`, and line 13 of `SignerRSA.sol`, "it's" should be "is" and "whose" should be "who's" or "who is".
- In line 35 of `SignerECDSA.sol`, line 38 of `SignerP256.sol`, and line 36 of `SignerRSA.sol`, "initializater" should be "initializer".
- In line 13 of `SignerERC7702.sol`, "ie." should be "i.e.".

In addition, the following instances were found in out-of-scope contracts:

- In line 24 of `draft-IERC4337.sol`, "on-chain contacts" should be "on-chain contracts".
- In line 167 and line 173 of `draft-IERC4337.sol`, "up" should be "upon".
- In line 203 of `draft-IERC7579.sol`, "onInstall" should be "onUninstall".

To improve the code readability, consider correcting the above instances of typographical errors.

Update: *Resolved in pull request #99 at commit ff0969a and in pull request #5561 at commit b498d9a.*

N-03 Misleading Pragma

The `AccountERC7579` contract has the following pragma: "pragma solidity ^0.8.24;". However, the code uses `require` with custom errors, which was introduced in Solidity 0.8.26. Therefore, this contract cannot be compiled with a version below 0.8.26.

Consider correcting the pragma to start from this version.

Update: *Resolved in pull request #99 at commit 94ba745. The Solidity Contracts team stated:*

> *Using 0.8.27 (in 0.8.26 the require with custom error is not available in the legacy pipeline).*

N-04 Unused Error

The `InvalidContentsType` error in `ERC7739Util.sol` is unused. Furthermore, the comment mentions `tryValidateContentsType`, which is not found in the codebase as well.

Consider removing this unused error and comment for clarity improved code clarity and maintainability.

Update: *Resolved in pull request #99 at commit 4fbed83.*

N-05 Use of Old Entrypoint

The `AccountCore` is a simple ERC-4337 compliant contract that depends on the Entrypoint contract. The `entryPoint` function uses version 0.7.

However, given that version 0.8 of the Entrypoint contract will be released soon, consider pointing to the address of the newer version or allowing the owner of the account to choose between using version 0.7 or 0.8.

Update: *Resolved in pull request #95 at commit 21d41ad and in pull request #97 at commit 7f8d84f.*

N-06 Unnecessary Processing

In the `decodeTypedDataSig` function of `ERC7739Utils.sol`, this line returns an empty string if `sigLength < 4`. However, if the signature length is less than 66, it will return an empty string later on in the function. That is because, at a minimum, 32 bytes are required for the separator, 32 bytes for the contents hash, and 2 bytes for the content description length.

Consider updating this check to return empty data if the `sigLength` is less than 66 in order to save gas in case of invalid signatures.

Update: *Resolved in pull request #99 at commit 3e2b528.*

N-07 Code Clarity Improvements

Throughout the codebase, multiple opportunities to improve code clarity were identified:

- When the `AccountERC7579Hooked` contract fails to install a hook module because there already is one installed, the error could be clearer and contain the already installed hook.
- If desired, the `accountId` function could be overridden by the `AccountERC7579Hooked` contract to display a different account name specifying that this account supports hooks, for example, "AccountERC7579Hooked".

Consider addressing the above instances to improve the clarity of the codebase.

Update: *Resolved in pull request #99 at commit 1e14e42 and at commit 43737d7.*

N-08 Non-strict Check on the Signature Length in `SignerP256.sol`

For the `_rawSignatureValidation` function in `SignerP256.sol`, if a signature is passed to the function that is longer than the signature length of 64 bytes, the function will just take the first 64 bytes and pass them to the verify function. This implies that a signature that is "valid" for the first 64 bytes, but has extra items attached at the end, would cause this function to return `true`. This is a design decision, motivated by giving freedom to the developer to use these extra arbitrary bytes.

However, the described behavior is not consistent across the other verifiers. Specifically, the implementation of `_rawSignatureValidation` in `SignerECDSA.sol` accepts the full signature and a strict check on the exact length of `65` bytes is performed in the called function from the Contracts library. Similarly, a strict check on the length of RSA signatures is performed in `RSA.sol`.

To ensure consistent behavior across all implemented verifying algorithms, consider modifying the check on the length in `SignerP256.sol` to be strict, as in `if (signature.length != 0x40) return false;`. Note that this still leaves developer the freedom to pass additional data in calldata after the signature by overriding the `_rawSignatureValidation` function of any of the three verifiers to add this functionality.

Update: *Acknowledged, not resolved. The Solidity Contracts team stated:*

> *While the recovery of P256 (secp256r1) signature does only use 64 bytes (r,s), some signers (like the one we use in our tests) do produce 65 bytes signature (r,s,v). The fact they verify doesn't use the v byte doesn't mean it should be an error to provide it. If provided, it can be safely ignored. If we were to modify the check, we would have to accept both 64 bytes long signature (completely used) or 65 bytes signatures (where the v byte is ignored). That would require a double condition, which is more expensive. In any way, we would be unable to distinguish between a 65th byte that is a v value, and something else. Overall, we think that it is ok to accept more than the required 64 bytes, and that anything more can be ignored. While we think it is essential to accept dropping 1 byte (for the v case), we decided to accept dropping any extra bytes provided. These extra bytes could be used by some extension to the signer. Our security doesn't rely on them, and we feel that their (eventual) presence doesn't weaken the security of the signer.*

Conclusion

The OpenZeppelin Community Contracts introduce additional contracts and features to the OpenZeppelin Contracts library, including smart accounts, `ERC20Bridgeable`, ERC-7739, and signers. We commend the Solidity Contracts team for addressing user needs by incorporating new standards, enhancing existing features, and adding new utilities.

During the audit, particular care was taken to document edge cases, ensuring that integrators are informed of potential risks when interacting with these contracts. Such efforts aim to create a more resilient codebase, recognizing the role of Community contracts as a foundational component within the blockchain ecosystem. The Contracts team has demonstrated a strong commitment to maximizing the security of these contracts and we are glad to have collaborated with them on this milestone.