OpenZeppelin | security

OpenZeppelin Contracts v5.5 Diff Audit

OpenZeppelin

October 17, 2025

Table of Contents

Summary

Type	Infrastructure
Timeline	From 2025-10-06 To 2025-10-09
Languages	Solidity

Total Issues	3 (3 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	2 (2 resolved)
Notes & Additional Information	1 (1 resolved)

Scope

OpenZeppelin performed a diff audit of the OpenZeppelin/openzeppelin-contracts repository, between base commit c64a1ed and target commit f5edfc0. This diff highlights all the changes made between the two commits.

In scope were the following files:

```
contracts
├──token/ERC20/utils/SafeERC20.sol
└──utils/cryptography
   ├──ECDSA.sol
   └──SignatureChecker.sol
```

System Overview

The scope only included the changes made to the `SafeERC20`, `ECDSA`, and `SignatureChecker` libraries:

- **`SafeERC20`**: The `_callOptionalReturn` function has been removed, and many of the functions that relied on it have been refactored. These refactors are intended to achieve the same functionality, but via the internal `_safeTransfer`, `_safeTransferFrom`, and `_safeApprove` functions, which are mostly implemented in assembly.

- **`ECDSA`**: Four new functions have been added: `tryRecoverCalldata`, `recoverCalldata`, `parse`, and `parseCalldata`. `tryRecoverCalldata` and `recoverCalldata` both mimic the already existing `memory` versions of themselves, called `tryRecover` and `recover`. The only difference is where the signature is stored (either `memory` or `calldata`). The `parse` and `parseCalldata` functions both take a dynamic-length signature as input and return the `v`, `r`, and `s` parameters for the signature.

- **`SignatureChecker`**: A new function, `isValidSignatureNowCalldata`, has been implemented. This function is a version of `isValidSignatureNow` that uses a `calldata signature` parameter. It also refactors the `isValidERC1271SignatureNow` function so that it is fully implemented in assembly.

Security Model and Trust Assumptions

During the audit, the following trust assumptions were made:

- The libraries are intended to be integrated as dependencies for other top-level contracts. It is assumed that they are used correctly as per the documentation within the contracts and the official OpenZeppelin docs.

- For the `ECDSA` and `SignatureChecker` libraries, it is assumed that users understand the risks of signature malleability, signature re-use, and the difference between 65-byte and 64-byte signatures.
- For the `SafeERC20` library, it is assumed that users understand ERC-20 compliance and have checked the tokens that are intended to be used with `SafeERC20` for compatibility. This is because `SafeERC20` implements extra restrictions on the allowed behavior.

Low Severity

L-01 Inconsistent `v` Normalization Between Signatures

The `parse` and `parseCalldata` helper functions split ECDSA signatures into the `v`, `r`, and `s` components for both 65-byte and EIP-2098 64-byte encodings. In the 64-byte path, `v` is derived from `vs` and normalized to `27` or `28`. In the 65-byte path, `v` is taken as-is. This yields inconsistent outputs for equivalent signatures: 65-byte inputs may return `v` which is `0` or `1`, while 64-byte inputs return `27` or `28`. Downstream code that expects canonical `v` can misbehave, and calls to `ecrecover` with `v` equal to `0` or `1` will return the zero address, potentially causing silent failures.

Consider normalizing `v` in the 65-byte branch to `27` or `28`, or removing the normalization from the 64-byte branch to be consistent with each other. In addition, consider updating the documentation to state that both helpers return the canonical `v`, `r` and `s` values that are suitable for `ecrecover`.

Update: *Resolved in pull request #5990. The OpenZeppelin Contracts team stated:*

> *Although the difference in the `v` value depending on whether the signature is 64-bytes or 65-bytes long may come across as an inconsistency, it's intentional:*
>
> *1. For 64-byte signatures: We must normalize because there's only 1 bit available (0 or 1), and ecrecover requires 27 or 28*
>
> *2. For 65-byte signatures: It should already be normalized by the signer. If v is 0 or 1, the signature is malformed and should fail cleanly when passed in to `tryRecover`*
>
> *The OpenZeppelin Contracts team included an improved NatSpec on the `parse` function to make the process clear.*

L-02 Incorrect Value in `isValidERC1271SignatureNow`

In the `SignatureChecker` library, within the `isValidERC1271SignatureNow` function, there is an incorrect hardcoded value. The comparison against `returndatasize()` checks if the return data is greater than `0x19` (or 25) bytes long, whereas it should ensure that the return data is greater than `0x1f` (or 31) bytes long. This check existed in the prior version.

In line 86 of the `SignatureChecker` library, consider changing `0x19` to `0x1f`.

Update: *Resolved in pull request #5973.*

Notes & Additional Information

N-01 Incorrect Comments in `isValidERC1271SignatureNow`

The inline comments documenting the calldata layout of the `isValidERC1271SignatureNow` function in memory misstate the 32-byte slot boundaries for the dynamic bytes argument. The comments show `[0x24 - 0x44]` for the signature offset and `[0x44 - 0x64]` for the signature length, implying 33-byte spans. However, the correct inclusive ranges should be `[0x24 - 0x43]` and `[0x44 - 0x63]` as both the signature offset and length are 32 bytes. While the code writes to the correct locations, inaccurate documentation can mislead maintainers and downstream implementations.

Consider correcting the comments to the exact ranges mentioned above and clarifying the fact that the ranges are inclusive.

Update: *Resolved in pull request #5959 at commit 7a4a7fe. The OpenZeppelin Contracts team stated:*

> *We fixed it by specifying the correct ranges.*

Conclusion

The audited scope included minimal changes, but they were made more complicated due to the extensive use of assembly. Many of the changes were re-implementations of already existing functions to leverage `calldata` for cheaper execution. Overall, the issues found in the codebase pertained to edge cases, but they should still be corrected given the wide-ranging use of OpenZeppelin Contracts libraries as dependencies. The changes were found to be well-thought-out and intentional, and did not break any existing functionality of the prior version of OpenZeppelin Contracts. The OpenZeppelin Contracts team is appreciated for their responsiveness and honesty when responding to questions about the audited codebase.