

ENIGMA DARK

Securing the Shadows



Security Review
Solady ERC6551

April, 2024

Enigma Dark Solady Code Review

Summary

Solady

Solady is a code library of hyper-optimized solidity snippets.

The contracts of Solady, hosted at [repo](#), were reviewed over 7 days, between April 1st and April 7th, 2024. The protocol was reviewed at commit [e4a14a5b365b353352f7c38e699a2bc9363d6576](#).

The scope of the security review consisted of the following contracts at the specific commit, ERC6551 and LibERC6551:

```
src/  
├── accounts  
│   ├── ERC6551.sol  
│   └── LibERC6551.sol
```

Disclaimer

This security review does not guarantee against a hack. It is a snapshot in time according to the specific commit. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, Solady and users of the contracts agree to use the code at their own risk. Any modifications to the code will require a new security review.

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

[L-01] - Precomputed result address in account is not cleaned up

Severity: Low Risk

Context: [LibERC6551.sol#L105](#)

Description: The address returned by the function `account` contains dirty upper bits. The return type of this variable does not in all cases cleanup this parameter (enforced by compiler). For instance, if the return value is subsequently used in another assembly block, this parameter might not be automatically cleaned up.

Recommendation: Consider cleaning up the upper bits of the returned address.

Vectorized: Our inline assembly dirty bits cleaning policy is as follows:

- For inputs, all unused bits must be cleaned. In the case of booleans, we will use `iszero(iszero(x))` to clean them.
- For outputs, we will clean the unused bits if it is at zero extra gas cost compared to a solution that does not clean the unused bits. But if it costs extra gas, we will leave the unused bits dirty. Regular Solidity will clean any unused bits on-the-fly.

[L-02] - Accounts could be unusable in case of hard fork

Severity: Low Risk

Context: [ERC6551.sol#L105](#)

Description: As per [ERC-6551](#), *"the inclusion of `chainId` as a parameter to `createAccount` allows the contract for a token bound account to be deployed at the same address on any supported chain"*. The `chainId` included in the account's bytecode is then always checked in important functions in the account, changing their behavior if the current chain ID is different from the chain ID inserted in account deployment. As an example, the `owner` function will return `address(0)` instead of the token owner if such scenario takes place:

```

function owner() public view virtual returns (address result) {
  /// @solidity memory-safe-assembly
  assembly {
    let m := mload(0x40) // Cache the free memory pointer.
    extcodecopy(address(), 0x00, 0x4d, 0x60)
    if eq(mload(0x00), chainid()) {
      ...
    }
    mstore(0x40, m) // Restore the free memory pointer.
  }
}

```

owner is then used inside the `_isValidSigner` internal function, which will be triggered in the `onlyValidSigner` modifier to ensure that the `execute/executeBatch` functions are called by the proper account signer. The problem with this approach is that in the event of a hard fork, the chain with the new chain ID will revert all relevant interactions with the account.

Recommendation: Remove the chain ID verification in the `owner` and `receiverFallback` functions, and update the parameter returned in the `token` function to return the actual `chainId` rather than the bytecode-stored id.

Vectorized: Fixed in [Vectorized/solady#884](#).

Informational Findings

[I-01] - Unused parameter can be removed to avoid silencing compiler warning

Severity: Informational

Context: [ERC6551.sol#L137](#)

Description: Unused function parameter should be commented out as a better and declarative way to silence runtime warning messages, instead of assigning the parameter to itself.

Recommendation: Update the `isValidSigner` function so that the `context` parameter is commented, and remove its self-assignment:

```

function isValidSigner(
    address signer,
-   bytes calldata context
+   bytes calldata /** context */
) public view virtual returns (bytes4 result) {
-   context = context; // Silence unused variable warning.
    bool isValid = _isValidSigner(signer);
    /// @solidity memory-safe-assembly
    assembly {
        // `isValid ? bytes4(keccak256("isValidSigner(address,bytes)")) : 0x00000000`.
        // We use `0x00000000` for invalid, in convention with the reference implementation.
        result := shl(224, mul(0x523e3260, iszero(iszero(isValid))))
    }
}

```

Vectorized: We prefer the variable = variable trick so that the presence of the named function argument can help with code auto-completion.