Coinbase SmartWallet

Security Review

Cantina Managed review by:
Optimum, Lead Security Researcher
Defsec, Security Researcher

April 23, 2024

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Gas Optimization . . . 4
  - 3.1.1 Change function visibility from `public` to `external` . . . 4
  - 3.1.2 The usage of the `[0:4]` operator in `CoinbaseSmartWallet` is redundant . . . 4
- 3.2 Informational . . . 4
  - 3.2.1 Simplify conditional execution . . . 4
  - 3.2.2 Potential issue with `WebAuthn` verification on unsupported networks . . . 5
  - 3.2.3 Consider implementing a `count()` function in `MultiOwnable` . . . 6
  - 3.2.4 Digital signatures used in the system will never expire . . . 7
  - 3.2.5 Inconsistent sorting of owners in account deployment . . . 7

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Smart Wallet is Coinbase's new ERC-4337 compliant smart contract wallet that supports multiple owners, passkey owners and Ethereum address owners, and cross-chain replayability for owner updates and other actions: sign once, update everywhere.

From Apr 15th to Apr 19th the Cantina team conducted a review of smart-wallet on commit hash 9edcf7f1. In addition, Solady's LibClone.sol was also in the review scope given its relevance. The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 0
- Gas Optimizations: 2
- Informational: 5

3 Findings

3.1 Gas Optimization

3.1.1 Change function visibility from `public` to `external`

Severity: Gas Optimization

Context: CoinbaseSmartWalletFactory.sol#L38

Description: The function `createAccount()` has been declared as `public`, but it is never called internally within the contract. It is best practice to mark such functions as `external` instead, as this saves gas. In cases where the function takes arguments, external functions can read the arguments directly from `calldata` instead of having to allocate `memory`.

```
function createAccount(bytes[] calldata owners, uint256 nonce)
        public
        payable
        virtual
        returns (CoinbaseSmartWallet account)
    {}
```

Recommendation: It's recommended to change the function visibility from `public` to `external`.

Coinbase: Addressed in PR 57.

Cantina Managed: Fixed in PR 57.

3.1.2 The usage of the `[0:4]` operator in `CoinbaseSmartWallet` is redundant

Severity: Gas Optimization

Context: CoinbaseSmartWallet.sol#L148, CoinbaseSmartWallet.sol#L184

Description: `CoinbaseSmartWallet` makes usage of the `[0:4]` operator to extract the first 4 bytes from `bytes calldata` type but as a matter of fact it is redundant since `bytes4` is already used in the same expression, as we can see in this example:

```
bytes4 selector = bytes4(call[0:4]);
```

Recommendation: Consider removing the usage of this operator in both instances mentioned in the context of this issue and only use the `bytes4` operator instead:

```
bytes4 selector = bytes4(call);
```

Coinbase: Addressed in PR 58.

Cantina Managed: Fixed in PR 58 by implementing the auditor's recommendation.

3.2 Informational

3.2.1 Simplify conditional execution

Severity: Informational

Context: CoinbaseSmartWalletFactory.sol#L53

Description: The current implementation uses an explicit comparison with false and an if statement to check if the account has already been deployed and initialize it if necessary. While this approach is correct and readable, it can be simplified further by leveraging the logical NOT operator (`!`):

```
function createAccount(bytes[] calldata owners, uint256 nonce)
    public
    payable
    virtual
    returns (CoinbaseSmartWallet account)
{
    if (owners.length == 0) {
        revert OwnerRequired();
    }

    (bool alreadyDeployed, address accountAddress) =
        LibClone.createDeterministicERC1967(msg.value, implementation, _getSalt(owners, nonce));

    account = CoinbaseSmartWallet(payable(accountAddress));

    if (alreadyDeployed == false) {
        account.initialize(owners);
    }
}
```

Recommendation: Consider simplifying the code by using NOT operator (`!`).

Coinbase: Addressed in PR 59.

Cantina Managed: Fixed in PR 59.

3.2.2 Potential issue with `WebAuthn` verification on unsupported networks

Severity: Informational

Context: CoinbaseSmartWallet.sol#L323

Description: The `WebAuthn` verification process employed in the provided code relies on the `FreshCryptoLib` library, which utilizes the precompiled `ModExp` contract (0x05) for efficient modular exponentiation. However, the availability and support for precompiled contracts can vary across different networks and clients.

While the main Ethereum network and most widely used clients support precompiled contracts, there is a possibility that some networks or clients may not have implemented or enabled the precompiled `ModExp` contract. This could lead to issues with the `WebAuthn` verification process on those unsupported networks or clients.

If the precompiled `ModExp` contract is not available or supported, the `FCL_nModInv` function, which performs modular inversion, will fail. Consequently, the `ecdsa_verify` function from FreshCryptoLib and the `WebAuthn.verify` function will also fail, effectively preventing successful `WebAuthn` authentication on those networks or clients.

```
function _validateSignature(bytes32 message, bytes calldata signature)
    internal
    view
    virtual
    override
    returns (bool)
{
    SignatureWrapper memory sigWrapper = abi.decode(signature, (SignatureWrapper));
    bytes memory ownerBytes = ownerAtIndex(sigWrapper.ownerIndex);

    if (ownerBytes.length == 32) {
        if (uint256(bytes32(ownerBytes)) > type(uint160).max) {
            // technically should be impossible given owners can only be added with
            // addOwnerAddress and addOwnerPublicKey, but we leave incase of future changes.
            revert InvalidEthereumAddressOwner(ownerBytes);
        }

        address owner;
        assembly ("memory-safe") {
            owner := mload(add(ownerBytes, 32))
        }

        return SignatureCheckerLib.isValidSignatureNow(owner, message, sigWrapper.signatureData);
    }

    if (ownerBytes.length == 64) {
        (uint256 x, uint256 y) = abi.decode(ownerBytes, (uint256, uint256));

        WebAuthn.WebAuthnAuth memory auth = abi.decode(sigWrapper.signatureData, (WebAuthn.WebAuthnAuth));

        return WebAuthn.verify({challenge: abi.encode(message), requireUV: false, webAuthnAuth: auth, x: x, y:
↪   y});
    }

    revert InvalidOwnerBytesLength(ownerBytes);
}
```

Recommendation: If certain networks or clients are identified as unsupported due to the lack of pre-compiled contract support and the fallback mechanism is not sufficient or feasible, it may be necessary to consider alternative authentication methods or libraries for those specific environments.

Coinbase: Acknowledged. We don't have immediate plans to launch on these networks and have also heard they plan to support EIP-7212.

Cantina Managed: Acknowledged.

3.2.3 Consider implementing a `count()` function in `MultiOwnable`

Severity: Informational

Context: MultiOwnable.sol#L124, MultiOwnable.sol#L141

Description: The current implementation of `MultiOwnable` already makes usage of the expression of `$.nextOwnerIndex - $.removedOwnersCount` in two different instances both in `removeOwnerAtIndex` and in `removeLastOwner`. This expression represents how many owners currently exist in the contract. Although using `$.nextOwnerIndex - $.removedOwnersCount` should be more efficient than calling a function that do the same calculation.

Recommendation: It might be useful to implement a dedicated public function for this expression so it can be used for external calls at the least.

Coinbase: Addressed in PR 60.

Cantina Managed: Fixed in PR 60 by implementing the auditor's recommendation.

3.2.4 Digital signatures used in the system will never expire

Severity: Informational

Context: CoinbaseSmartWallet.sol#L323

Description: The provided code snippet, `_validateSignature`, is responsible for validating signatures in the context of a smart contract. In the current version of the code there is no mechanism in place to invalidate a specific digital signatures after expiry which is a common practice in many implementations. The only workaround is to completely remove the owner that signed the transaction or change its index.

Setting an expiration date for digital signatures is important in volatile markets where the trend may shift quickly, and also to limit the risk of an exploit in case the digital signature algorithm is compromised.

```
function _validateSignature(bytes32 message, bytes calldata signature)
    internal
    view
    virtual
    override
    returns (bool)
{
    SignatureWrapper memory sigWrapper = abi.decode(signature, (SignatureWrapper));
    bytes memory ownerBytes = ownerAtIndex(sigWrapper.ownerIndex);

    if (ownerBytes.length == 32) {
        if (uint256(bytes32(ownerBytes)) > type(uint160).max) {
            // technically should be impossible given owners can only be added with
            // addOwnerAddress and addOwnerPublicKey, but we leave incase of future changes.
            revert InvalidEthereumAddressOwner(ownerBytes);
        }

        address owner;
        assembly ("memory-safe") {
            owner := mload(add(ownerBytes, 32))
        }

        return SignatureCheckerLib.isValidSignatureNow(owner, message, sigWrapper.signatureData);
    }

    if (ownerBytes.length == 64) {
        (uint256 x, uint256 y) = abi.decode(ownerBytes, (uint256, uint256));

        WebAuthn.WebAuthnAuth memory auth = abi.decode(sigWrapper.signatureData, (WebAuthn.WebAuthnAuth));

        return WebAuthn.verify({challenge: abi.encode(message), requireUV: false, webAuthnAuth: auth, x: x, y:
↪   y});
    }

    revert InvalidOwnerBytesLength(ownerBytes);
}
```

Recommendation: To mitigate the risk of never expired signatures and enhance the security of the system, it is recommended to include a deadline parameter in the signature verification process.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.2.5 Inconsistent sorting of owners in account deployment

Severity: Informational

Context: CoinbaseSmartWalletFactory.sol#L38

Description: The current implementation of the `createAccount` function in the provided code does not enforce a specific order for the owners provided in the owners array. This can lead to potential issues or inconsistencies when creating multiple accounts with the same set of owners.

The `test_orderOfOwners` function demonstrates this behavior. It creates two accounts using the same set of owners but in different orders (`ownersWithSorted` and `ownersWithRandom`). While the deployment addresses of these accounts are deterministic based on the set of owners and the nonce, the order of the owners is not enforced:

```
function test_orderOfOwners() public {
    ownersWithSorted.push(abi.encode(address(1)));
    ownersWithSorted.push(abi.encode(address(2)));
    ownersWithSorted.push(abi.encode(address(3)));
    ownersWithSorted.push(abi.encode(address(4)));
    ownersWithSorted.push(abi.encode(address(5)));
    address preDetermined = factory.getAddress(ownersWithSorted, 0);
    CoinbaseSmartWallet a = factory.createAccount{value: 1e18}(ownersWithSorted, 0);

    ownersWithRandom.push(abi.encode(address(1)));
    ownersWithRandom.push(abi.encode(address(3)));
    ownersWithRandom.push(abi.encode(address(4)));
    ownersWithRandom.push(abi.encode(address(2)));
    ownersWithRandom.push(abi.encode(address(5)));


    address randomOrder = factory.getAddress(ownersWithRandom, 0);
    CoinbaseSmartWallet b = factory.createAccount{value: 1e18}(ownersWithRandom, 0);
}
```

Output:

```
[PASS] test_orderOfOwners() (gas: 1445896)
Traces:
  [1445896] CoinbaseSmartWalletFactoryTest::test_orderOfOwners()
    [3144] CoinbaseSmartWalletFactory::getAddress([0x0000000000000000000000000000000000000000000000000000000000000001
    ↪ 00000001, 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005], 0) [staticcall]
      ← [Return] 0xe6a962bFc8414f5359375B5F2cd0E364063C8E04
    [456232] CoinbaseSmartWalletFactory::createAccount{value:
    ↪ 100000000000000000}([0x0000000000000000000000000000000000000000000000000000000000000001,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005], 0)
      [34337] → new <unknown>@0xe6a962bFc8414f5359375B5F2cd0E364063C8E04
        ← [Return] 61 bytes of code
      [381594] 0xe6a962bFc8414f5359375B5F2cd0E364063C8E04::initialize([0x0000000000000000000000000000000000
    ↪ 00000000000000000000000001, 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005])
        [378734]
    ↪ CoinbaseSmartWallet::initialize([0x0000000000000000000000000000000000000000000000000000000000000001,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005]) [delegatecall]
          emit AddOwner(index: 0, owner: 0x0000000000000000000000000000000000000000000000000000000000000001
)
          emit AddOwner(index: 1, owner: 0x0000000000000000000000000000000000000000000000000000000000000002
)
          emit AddOwner(index: 2, owner: 0x0000000000000000000000000000000000000000000000000000000000000003
)
          emit AddOwner(index: 3, owner: 0x0000000000000000000000000000000000000000000000000000000000000004
)
          emit AddOwner(index: 4, owner: 0x0000000000000000000000000000000000000000000000000000000000000005
)
          ← [Stop]
        ← [Return]
      ← [Return] 0xe6a962bFc8414f5359375B5F2cd0E364063C8E04
    [3144] CoinbaseSmartWalletFactory::getAddress([0x0000000000000000000000000000000000000000000000000000000000000001
    ↪ 00000001, 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005], 0) [staticcall]
      ← [Return] 0xeb744BE8b9852e4a0666A8255C21c7EbC35054Ce
    [453732] CoinbaseSmartWalletFactory::createAccount{value:
    ↪ 100000000000000000}([0x0000000000000000000000000000000000000000000000000000000000000001,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005], 0)
      [34337] → new <unknown>@0xeb744BE8b9852e4a0666A8255C21c7EbC35054Ce
        ← [Return] 61 bytes of code
      [379094] 0xeb744BE8b9852e4a0666A8255C21c7EbC35054Ce::initialize([0x0000000000000000000000000000000000
    ↪ 00000000000000000000000001, 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005])
        [378734]
    ↪ CoinbaseSmartWallet::initialize([0x0000000000000000000000000000000000000000000000000000000000000001,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000003,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000004,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000002,
    ↪ 0x0000000000000000000000000000000000000000000000000000000000000005]) [delegatecall]
          emit AddOwner(index: 0, owner: 0x0000000000000000000000000000000000000000000000000000000000000001
)
          emit AddOwner(index: 1, owner: 0x0000000000000000000000000000000000000000000000000000000000000003
)
          emit AddOwner(index: 2, owner: 0x0000000000000000000000000000000000000000000000000000000000000004
)
          emit AddOwner(index: 3, owner: 0x0000000000000000000000000000000000000000000000000000000000000002
)
          emit AddOwner(index: 4, owner: 0x0000000000000000000000000000000000000000000000000000000000000005
)
          ← [Stop]
        ← [Return]
      ← [Return] 0xeb744BE8b9852e4a0666A8255C21c7EbC35054Ce
    ← [Stop]
```

This inconsistent sorting of owners can cause problems in scenarios where the order of owners matters, such as when performing multi-signature operations or when calculating deterministic addresses based on the order of owners.

Recommendation: To ensure consistent behavior and avoid potential issues, it is recommended to enforce a specific sorting order for the owners when deploying accounts.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.