CANTINA

Ethereum Foundation: Account Abstraction Infrastructure

Security Review

Cantina Managed review by:

Kaden, Lead Security Researcher
Gerard Persoon, Lead Security Researcher

November 16, 2025

Contents

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 level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

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 are 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

The Ethereum Foundation is a non-profit organization that supports the Ethereum ecosystem.

From Nov 2nd to Nov 3rd the Cantina team conducted a review of account-abstraction on commit hash 86fcd84c. The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	8	7	1
Total	**10**	**9**	**1**

The Cantina Managed team reviewed Ethereum Foundation's `account-abstraction` holistically on commit hash f54584ed and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 Result of `Paymaster::validatePaymasterUserOp()` not sufficiently checked

Severity: Low Risk

Context: EntryPoint.sol#L224, EntryPoint.sol#L248-L251, EntryPoint.sol#L669, EntryPoint.sol#L675-L678, EntryPoint.sol#L686-L711

Description: Function `_callValidatePaymasterUserOp()` calls `Paymaster::validatePaymasterUserOp()`, but doesn't check the result sufficiently. For example if 0 bytes are returned, `validationData` and `context` retrieve a value from memory that isn't initialized and thus uses a value that happens to be there. See the first proof of concept below.

Additonally the check `contextLength + 31 < maxContextLength` doesn't seem logical. This check enforces that `contextLength` is larger than or equal to `maxContextLength - 31`. However normally a check is done that the received data is smaller than a maximum. This allows `contextLength` to be very large, which will use a lot of gas for memory expansion at the moment where context is used in `abi.encodeCall()` within `_executeUserOp()`. See the second POC below.

Furthermore, due to the `unchecked`, `contextLength + 31` can wrap to a number smaller than 31, if `contextLength` is close to `type(uint256).max`.

If `context` is very large then the code will revert in `abi.encodeCall()` within `_executeUserOp()`. If `context` is reasonably large it can consume a lot of gas due to memory expansion.

Proof of concept:

- Proof of concept 1 (len is 0): Here is a proof of concept, where `len` is 0, that can be run in Remix:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.28;
import "hardhat/console.sol";

contract testpay {
function validatePaymasterUserOp() external {
    } // returns 0 bytes
function _getFreePtr() internal pure returns (uint256 ptr) {
        assembly ("memory-safe") { ptr := mload(0x40) }
    }
function test() public {
        uint256 freePtr = _getFreePtr();
        bytes memory validatePaymasterCall = abi.encodeCall(testpay.validatePaymasterUserOp,());
        bytes memory context;
        uint256 validationData;
        address paymaster = address(this);
        uint256 paymasterVerificationGasLimit = gasleft();
        bool success;
        uint256 contextLength;
        uint256 contextOffset;
        uint256 maxContextLength;
        uint256 len;
        assembly ("memory-safe") {
            success := call(paymasterVerificationGasLimit, paymaster, 0, add(validatePaymasterCall,
            ↪  0x20), mload(validatePaymasterCall), 0, 0)
            len := returndatasize()
            returndatacopy(freePtr, 0, len)
            validationData := mload(add(freePtr, 32))
            contextOffset := mload(freePtr)
            maxContextLength := sub(len, 96)
            context := add(freePtr, 64)
            contextLength := mload(context)
        }
        console.log(len); // 0
        console.log(maxContextLength); //
        ↪  1157920892373161954235709850086879078532699846656405640394575840079131296398 40
        console.log(validationData); //
        ↪  3406535597288997965013918829336121169467386632844212440960811337274513319526 4
        console.log(context.length); //
        ↪  9966937589691808071229720261006577406076338010222259230285550678179 84
```

```
    }
}
```

- Proof of concept 2 (`contextLength` is large): This proof of concept shows what happens when `contextLength` is large. The check `contextLength + 31 < maxContextLength` doesn't block this.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.28;
import "hardhat/console.sol";

contract testpay {
    function TestCallValidatePaymasterUserOp(uint n) public returns (uint) {
        bytes memory context;
        assembly ("memory-safe") { mstore(context,n) }
        uint maxContextLength = 0;
        uint contextLength = context.length;
        if (contextLength + 31 < maxContextLength) {revert ("maxlen"); }
        uint g1=gasleft();
        bytes memory encoded = abi.encode(context);
        uint g2=gasleft();
        return (g1-g2);
    }
    constructor() {
        uint n=1;
        for (uint i=0; i< 10;i++) {
            uint g = TestCallValidatePaymasterUserOp(n);
            console.log(i,n,g);
            n *= 10;
        }
    }
}
```

This has the following output when run in Remix:

```
0 1 470
1 10 467
2 100 486
3 1000 659
4 10000 2622
5 100000 43116
6 1000000 2525909
7 10000000 235071808
```

Recommendation: Consider changing the checks to:

```
- if (!success || contextOffset != 64 || contextLength + 31 < maxContextLength) { revert(...); }
+ if (!success ||  len < 96 || contextOffset != 64 ||
+    ( contextLength + 31 ) >> 5 ) << 5 != maxContextLength) { revert(...); }
```

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Verified.

3.1.2 Can't specify `validAfterBlock == 0`

Severity: Low Risk

Context: EntryPoint.sol#L764-L774

Description: It is not possible to specify `validAfterBlock == 0`, because that would be `validAfter == 0x800000000000`. However that value won't pass the test `data.validAfter > VALIDITY_BLOCK_RANGE_FLAG`. This would result in the situation where the code would switch to time ranges and would interpret the `validUntil` as a (very large) time value.

Recommendation: Consider changing the code to:

```
- if (data.validAfter > VALIDITY_BLOCK_RANGE_FLAG && data.validUntil > VALIDITY_BLOCK_RANGE_FLAG) {
+ if (data.validAfter >= VALIDITY_BLOCK_RANGE_FLAG && data.validUntil >= VALIDITY_BLOCK_RANGE_FLAG) {
```

Alternatively the following can be used:

```
if ((data.validAfter & VALIDITY_BLOCK_RANGE_FLAG) != 0 && (data.validUntil & VALIDITY_BLOCK_RANGE_FLAG) != 0) {
```

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 Incorrect `initCode` comment

Severity: Informational

Context: PackedUserOperation.sol#L23-L24

Description: In `PackedUserOperation.sol`, we include the following comment documenting the behavior of providing `initCode`:

```
/*
 * - initCode:
 *     * non-7702: `initCode = factory(20) || factoryCalldata`; the factory must return `sender` and deploy code.
 */
```

As can be seen in the comment, it's noted that the factory must deploy code if `initCode` is provided. However, this is no longer true with the recent change to `EntryPoint._createSenderIfNeeded` in which we return early if the `sender` already has code:

```
if (sender.code.length != 0) {
    // ignoring the initcode for an existing 'sender' contract
    emit IgnoredInitCode(
        opInfo.userOpHash,
        sender,
        factory
    );
    return;
}
```

Recommendation: Adjust the comment to indicate that the factory may deploy code:

```
  /*
   * - initCode:
-  *     * non-7702: `initCode = factory(20) || factoryCalldata`; the factory must return `sender` and deploy
↪   code.
+  *     * non-7702: `initCode = factory(20) || factoryCalldata`; the factory must return `sender` and may
↪   deploy code.
```

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Verified.

3.2.2 Incorrect memory comment

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In `Helpers.calldataKeccakWithSuffix`, we include the following comment documenting the function behavior:

```
/*
 * But more efficient, and doesn't leave the copied data in memory.
```

However, the copied data actually is left in memory. As we can see in the below snippet, we `calldatacopy` and `mstore` to update memory before executing a `keccak256` operation:

```
function calldataKeccakWithSuffix(bytes calldata data, uint256 len, bytes8 suffix) pure returns (bytes32 ret) {
    assembly ("memory-safe") {
        let mem := mload(0x40)
        calldatacopy(mem, data.offset, len)
        mstore(add(mem, len), suffix)
        len := add(len, 8)
        ret := keccak256(mem, len)
    }
}
```

Since we copy data to free memory and `keccak256` uses but does not consume this memory, the copied data is left in memory.

Recommendation: Adjust the comment to indicate that the copied data is left in memory:

```
- * But more efficient, and doesn't leave the copied data in memory.
+ * But more efficient, although it leaves the copied data in memory.
```

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.

3.2.3 Typos/misspellings

Severity: Informational

Context: Eip7702Support.sol#L78, EntryPoint.sol#L808, IEntryPointSimulations.sol#L65, IEntryPoint.sol#L130, IEntryPoint.sol#L151, INonceManager.sol#L20, PackedUserOperation.sol#L25

Description: The following typos/misspellings have been noted in the code:

- `Eip7702Support:L78`: "if some info" is not clear.
- `EntryPoint.sol:L808`: "no required" should be "not required".
- `IEntryPointSimulation.sol:L65`: "the the success" should be "the success":
- `IEntryPoint.sol:L130`: "andhandleAggregatedOps" should be "and handleAggregatedOps".
- `IEntryPoint.sol:L151`: "cought" should be "caught".
- `INonceManager.sol:L20`: ".." should be ".".
- `PackedUserOperation.sol:L25`: "initizlizationCode" should be "initializationCode".

Recommendation: Fix the listed typos/misspellings as recommended above.

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.

3.2.4 `validUntil == 0` can also be used for blockranges

Severity: Informational

Context: EntryPoint.sol#L764-L781, Helpers.sol#L46-L53

Description: The documentation for erc-4337 contains the following:

> `validUntil` is 6-byte timestamp value, or zero for "infinite". The `UserOperation` is valid only up to this time. In order to specify a validity range using block numbers, both the `validUntil` and `validAfter` need to set their highest bit to 1.
> *Note: The validity range can be expressed by two block timestamps or two block numbers, but one timestamp and one block number cannot be mixed in the same UserOperation's validity range*.

In the code, when `validUntil == 0`, it will be replaced with `type(uint48).max`, which has all bits set. The result is that it can both be used as very large timestamp and as a very large blocknumber. This seems useful and might be intentional. However the documentation isn't clear about this situation.

Recommendation: Doublecheck if using `validUntil == 0` for blockranges is also desired. If so, also include this in the documentation.

Ethereum Foundation: Acknowledged. We will document this as a feature.

Cantina Managed: Acknowledged.

3.2.5 `SPDX` not on first line

Severity: Informational

Context: Eip7702Support.sol#L1-L2

Description: The `SPDX` in `Eip7702Support` isn't on the first line. According to the solidity suggested layout source file should start with it.

Recommendation: Consider moving the `SPDX` statement to the top of the file.

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.

3.2.6 Not all functions are `virtual`

Severity: Informational

Context: Simple7702Account.sol#L40, Simple7702Account.sol#L60, SimpleAccountFactory.sol#L33, SimpleAccountFactory.sol#L55, SimpleAccount.sol#L102, SimpleAccount.sol#L109, SimpleAccount.sol#L118, NonceManager.sol#L17-L18

Description: Several functions have been updated to be `virtual`, however several other function are not `virtual` yet.

Recommendation: Consider changing all relevant functions to `virtual`.

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.

3.2.7 Virtual function not always used

Severity: Informational

Context: SimpleAccount.sol#L72-L75, BasePaymaster.sol#L109-L111, BasePaymaster.sol#L118-L123, BasePaymaster.sol#L135-L136

Description: In several places there are virtual functions that can be used to access storage variables. However sometimes the storage variables are access directly. This can cause issues when the virtual functions are overridden.

Recommendation: Consider using the virtual functions where possible.

```
- _entryPoint
+ entryPoint()
```

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.

3.2.8 `currentUserOpHash` isn't cleared

Severity: Informational

Context: EntryPoint.sol#L76-L94, EntryPoint.sol#L97-L100, EntryPoint.sol#L153, EntryPoint.sol#L224-L232

Description: The erc-4337 documentation says:

```
// get the currently executing UserOperation hash, or 0 if not called during the execution
function getCurrentUserOpHash() public view returns (bytes32);
```

However `currentUserOpHash` is never cleared and in the call to `_compensate()`, which does an external call, `getCurrentUserOpHash()` still retrieves the last `UserOpHash`, while it should be 0 according to the documentation.

Note: the `nonReentrant` modifier prevents other execution outside of `handleOps()` and `handleAggregatedOps()`.

Recommendation: Consider cleaning `currentUserOpHash` before (or in) the calls to `_compensate()` or update the documentation.

Ethereum Foundation: Fixed in commit 8bc437da.

Cantina Managed: Fix verified.