EIP-4337 – Ethereum Account Abstraction Incremental Audit

OPENZEPPELIN SECURITY | MARCH 1, 2023

Security Audits

April 12, 2023

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

Type
: DeFi

Timeline
: From 2023-01-09

Total Issues
27 (23 resolved, 4 partially resolved)
Critical Severity Issues
0 (0 resolved)
High Severity Issues
1 (1 resolved)
Medium Severity Issues
0 (0 resolved)
Low Severity Issues
12 (10 resolved, 2 partially resolved)
Notes & Additional Information
14 (12 resolved, 2 partially resolved)

Scope

EIP-4337 is a specification to add account abstraction functionality to the Ethereum mainnet without modifying the consensus rules. The Ethereum Foundation asked us to review the latest version revision of their specification and reference implementation.

We audited the eth-infinitism/account-abstraction repository at the 6dea6d8752f64914dd95d932f673ba0f9ff8e144 commit.

In scope were the following contracts:

```
contracts
├── bls
│   ├── BLSAccount.sol
│   ├── BLSAccountFactory.sol
│   ├── BLSSignatureAggregator.sol
│   └── IBLSAccount.sol
├── core
│   ├── BaseAccount.sol
│   ├── BasePaymaster.sol
│   ├── EntryPoint.sol
│   ├── SenderCreator.sol
│   └── StakeManager.sol
├── gnosis
│   ├── EIP4337Fallback.sol
```

```
|   ├── IAccount.sol
|   ├── IAggregatedAccount.sol
|   ├── IAggregator.sol
|   ├── ICreate2Deployer.sol
|   ├── IEntryPoint.sol
|   ├── IPaymaster.sol
|   ├── IStakeManager.sol
|   └── UserOperation.sol
├── samples
|   ├── DepositPaymaster.sol
|   ├── IOracle.sol
|   ├── SimpleAccount.sol
|   ├── SimpleAccountFactory.sol
|   ├── TestAggregatedAccount.sol
|   ├── TestAggregatedAccountFactory.sol
|   ├── TestSignatureAggregator.sol
|   ├── TokenPaymaster.sol
|   └── VerifyingPaymaster.sol
└── utils
    └── Exec.sol
```

Originally `BLSHelper.sol` was in scope, but we agreed to deprioritize a complete review during the audit.

Update

After the audit, the Ethereum Foundation asked us to review three new pull requests:

- Pull Request #245 merged at commit `1b85cfb`: creates a canonical structure for the user operation hash preimage to prevent possible hash collisions between different user operations.
- Pull Request #247 merged at commit `19918cd`: moves nonce uniqueness validation to the `EntryPoint` contract. This now prevents accounts from reusing a nonce across multiple operations, but the new “key” and “sequence number” distinction provides some flexibility with operation ordering.

event-ordering confusion that could occur if the functions were called recursively.

As part of the fix review process and our review of these changes, we reviewed the pull requests that affect in-scope contracts up to commit `9b5f2e4`.

System Overview

The system architecture is described in our original audit report, and now contains a series of important changes.

For instance, users and paymasters can now both change the EVM state when validating an operation. This is more general and mitigates the need for a paymaster to have after-revert functionality, which may be removed in a future version. To support this change, additional storage restrictions (described in the EIP) have been added to ensure all validations in a batch access non-overlapping sets of storage slots. In addition, user operations can delegate their validation to an "Aggregator" smart contract, which allows all operations that share an aggregator to be validated together. Aggregators are subject to the same staking and throttling rules as paymasters.

To forestall possible confusion, it is worth noting that in this context, "aggregation" refers to any mechanism that can authenticate independent user operations efficiently. The sample `BLSSignatureAggregator` contract efficiently validates several BLS signatures over different user operations, but does not use the standard BLS Signature Aggregation technique, which produces a combined signature over a single message. Regardless, the system supports accounts with arbitrary validation logic, so anyone could deploy an account that accepts aggregate BLS signatures over a single message (to produce a multi-signature wallet, for example).

There are also a few incremental changes:

- New accounts are now initialized with user-chosen factory contracts to provide more flexibility during deployment.
- The term "wallet" has been replaced with "account".
- Users can set time restrictions that define when an operation is valid.
- Senders can now have multiple operations in a batch if they are also staked.

Client reported: *The Ethereum Foundation identified this issue during the audit.*

During simulation, the `EntryPoint` contract invokes a view function on the sender contract, before proceeding with the regular validation. Since the first access of any storage slot is more expensive than subsequent accesses, the view function could perform the initial "cold accesses" to allow the regular validation function to use "warm accesses". If the different gas costs determined whether the validation function ran out of gas, the validation would succeed during simulation but fail on-chain. In this scenario, the bundler would have to pay for the failed transaction.

Update: *Resolved in pull request #216 and merged at commit `1f505c5`. The aggregator logic has been redesigned, which makes this issue obsolete.*

Leaked Base Fee

Client reported: *The Ethereum Foundation identified this issue before the audit.*

The EIP forbids accounts from using the `BASEFEE` opcode during validation, to prevent them from detecting when they are being simulated offline. However, the `EntryPoint` contract passes the required pre-fund to the account, which depends on the base fee, thereby leaking this value.

Update: *Resolved in pull request #171 and merged at commit `b34b7a0`. The prefund amount now uses the maximum possible gas price.*

Replay on verifying paymaster

Client reported: *The Ethereum Foundation shared this issue with us during the audit after it was reported by leekt.*

The `VerifyingPaymaster` contract requires the trusted signer to sign a hash of a user operation. However, the signature is under-specified. In particular:

- It is not locked to a particular chain or paymaster.
- It does not take advantage of the new time restriction option.

Update: *Resolved in pull request #184 and merged at commit* `48854ef`.

Self-destruct `EIP4337Manager`

Client reported: *The Ethereum Foundation shared this issue with us during the audit after it was reported by leekt.*

The `EIP4337Manager` contract is intended to augment `GnosisSafe` contracts, by providing a user op validation function. Safe contracts (technically their proxies) are intended to use `delegatecall` to access this function.

However, anyone can configure the manager contract with new modules. Since the manager contract inherits `GnosisSafe` functionality, the new modules can trigger arbitrary function calls and potentially self-destruct the contract. This would effectively disable the manager module for all safes that used it.

Update: *Resolved in pull request #208 and merged at commit* `d92fec8`.

Revert reason bombing

Client reported: *The Ethereum Foundation identified this issue during the audit.*

The `EntryPoint` contract has four locations where an external function call can revert with an arbitrarily large message that the `EntryPoint` must copy to its own memory. Each instance has a different practical consequence:

- The first instance occurs after a user operation completes, which effectively allows the operation to consume much more than the allocated `callGasLimit`. If this occurs on-chain, the user (or the paymaster) will still be charged for the extra gas consumed. If instead the entire bundle reverted, the `FailedOp` error would not be returned, so the bundler would not easily recognize the problematic operation.
- The second instance occurs when a user's validation fails. This operation will be discarded anyway without being added to a bundle, so it can be ignored.
- The third and fourth instances occur when the paymaster validates or concludes a user operation. Either could occur for the first time once the operation is in a bundle, and could

operation revert reason was limited.

High Severity

Invalid aggregate signature [samples]

The `BLSSignatureAggregator` exposes a mechanism to let the bundler validate individual signatures before constructing the bundle. Successful operations are grouped so the bundler can combine their signatures off-chain and the `EntryPoint` can validate them together on-chain. However, it is possible for an account to construct an operation that will pass the individual-signature check and still fail the combined-signature check.

In particular, if the public key it exposes during the individual validation is different from the one used during the combined validation, the two validations will be inconsistent even though the signature is the same. This could occur if the last 4 words of the `initCode` do not match the public key (because the `initCode` has additional data, or if they do not use the expected creation function). It could also occur if the user's validation function (which is not invoked during the individual signature validation) changes the public key that is returned by `getBlsPublicKey`.

If a bundler constructs a bundle with these operations, it will be unable to validate the combined signature and will attribute the fault to the aggregator, which will cause the aggregator to be throttled and user operations with the same aggregator will not be processed.

Consider synchronizing the two validation functions so they both use the same public key.

Update: *Resolved in pull request #195 as well as commit `268f103` of pull request #216, which were merged at commits `1cc1c97` and `1f505c5` respectively.*

Low Severity

Accounts cannot replace `EntryPoint` [samples]

The comments describing the `initialize` function of the `SimpleAccount` contract claim there should be a mechanism to replace the `EntryPoint` contract. This does not match the

Consider updating the comment to match the behavior, and introducing a mechanism to replace the `EntryPoint` contract if that functionality is desired.

Update: *Resolved in pull request #192 and merged at commit `82685b2`. A @dev comment was added to the docstring of the `initialize` function to clarify that the `_entryPoint` storage variable is not a parameter of the initializer because an upgrade is required to change the EntryPoint address.*

Gnosis safe reverts on signature failure [samples]

The documentation for the `SIG_VALIDATION_FAILED` constant states that `validateUserOp` must return this value instead of reverting if signature validation fails. The `SimpleAccount` contract correctly follows the specification, however in the `EIP4337Manager` contract, the `validateUserOp` function reverts if the signature validation fails. This means the `simulateValidation` function will revert without providing a `ValidationResult` object.

Consider changing the logic so that `validateUserOp` returns `SIG_VALIDATION_FAILED` in all cases where an invalid signature is encountered.

Update: *Resolved in pull request #181 and merged at commit `1dfb173`.*

Imprecise time range [core]

The `EntryPoint` contract decrements the operation expiry timestamp in order to convert `0` (which should be interpreted as “no expiry”) to the maximum `uint64` value. However, every other possible expiry value is now off by one. In the interest of predictability, consider only modifying the `0` timestamp.

Update: *Resolved in pull request #193 and merged at commit `973c0ac`.*

Incorrect or misleading documentation [core and samples]

Several docstrings and inline comments throughout the code base were found to be incorrect or misleading. In particular:

- In `BLSSignatureAggregator.sol`:
  - Line 117: The docstring references a call to `simulateUserOperation`. The function name should be `simulateValidation`.
- In `EIP4337Manager.sol`:
  - Line 21: The docstring states the contract inherits `GnosisSafeStorage`, but it actually inherits `GnosisSafe`.
- In `EntryPoint.sol`:
  - Line 180: The comment does not include `paymasterAndData` as one of the dynamic byte arrays being excluded from `MemoryUserOp`.
  - Line 393: The docstring states that `_validatePaymasterPrepayment` validates that the paymaster is staked, but the function does not perform this check.
- In `IPaymaster.sol`:
  - Lines 25-26: The docstring states that the `validUntil` and `validAfter` timestamps are 4 bytes in length, but these are 8-byte (uint64) values.
- In `IStakeManager.sol`:
  - Line 7, lines 43-44: Docstrings in this contract refer to staking only for paymasters, implying this is the only entity that should stake. Signature aggregators and factories are also required to stake following the same rules as paymasters.
  - Line 45: The docstring makes a reference to the "global unstakeDelaySec", which no longer exists.
  - Line 47: The `DepositInfo` docstring explains that the variable sizes were chosen so that `deposit` and `staked` fit into a single `uint256` word, but the 3rd parameter `stake` will also fit.
- In `SimpleAccount.sol`:
  - Line 52: The comment makes a reference to the `execFromEntryPoint` function, which no longer exists.
  - Line 57: The docstring for `execute` says "called directly from owner, not by entryPoint", but the `_requireFromEntryPointOrOwner` function allows `execute` to be called by the EntryPoint. The comment isn't clear on whether it is a suggestion, or a restriction to be enforced.
  - Lines 75-79: The docstring does not match the `initialize` function.

    - Line 26: The `@success` parameter is listed in the wrong order.
- In `UserOperation.sol`:
    - Line 25: The `callGasLimit` parameter has no `@param` statement.

Update: *Resolved in pull request #194 and pull request #216, which were merged at commits `faf305e` and `1f505c5` respectively.*

Misleading specification [core]

The EIP states that when a `FailedOp` is detected, all other operations from the same paymaster should be removed from the current batch. However, this should only apply to `FailedOp` errors that explicitly mention the paymaster, which imply the paymaster was at fault. Operations that fail for unrelated reasons should not penalize their paymaster.

The EIP also states that `userOp` validation cannot call the `handleOps` method. This restriction should also apply to `handleAggregatedOps`.

Consider clarifying these points in the EIP.

Update: *Partially resolved in pull request #196 and merged at `5929ff8`. The updated EIP mistakenly refers to the EntryPoint's `depositTo` function as `depositFor`.*

Mismatched event parameter [core]

The `StakeLocked` event specifies a `withdrawTime` parameter, but the argument passed in is the new unstake delay. Consider renaming the event parameter to match its actual usage.

Update: *Resolved in pull request #197 and merged at commit `545a15c`.*

Missing docstrings [core and samples]

Throughout the codebase there are several parts that do not have docstrings. For instance:

- Line 24 in `BLSAccount.sol`
- Line 39 in `BLSAccount.sol`
- Line 44 in `BLSAccount.sol`

- Line 106 in `BLSSignatureAggregator.sol`
- Line 10 in `IBLSAccount.sol`
- Line 24 in `BasePaymaster.sol`
- Line 29 in `BasePaymaster.sol`
- Line 31 in `BasePaymaster.sol`
- Line 167 in `EntryPoint.sol`
- Line 18 in `StakeManager.sol`
- Line 11 in `EIP4337Fallback.sol`
- Line 23 in `GnosisAccountFactory.sol`
- Line 67 in `IStakeManager.sol`
- Line 34 in `UserOperation.sol`
- Line 73 in `DepositPaymaster.sol`
- Line 27 in `SimpleAccount.sol`
- Line 31 in `SimpleAccount.sol`
- Line 23 in `TestAggregatedAccount.sol`
- Line 34 in `TestAggregatedAccount.sol`
- Line 16 in `TestSignatureAggregator.sol`
- Line 28 in `TestSignatureAggregator.sol`
- Line 43 in `TestSignatureAggregator.sol`
- Line 40 in `TokenPaymaster.sol`
- Line 6 in `Exec.sol`

Consider thoroughly documenting all functions and their parameters, especially public APIs. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #212 and merged at commit `eeb93b2`. The recommended changes to `GnosisAccountFactory.sol` were not implemented.*

Missing error messages in require statements [core and samples]

Within the codebase there are some `require` statements that lack error messages:

- The `require` statement on line 105 of `BasePaymaster.sol`

Consider including specific, informative error messages in `require` statements to improve overall code clarity and facilitate troubleshooting whenever a requirement is not satisfied.

Update: *Resolved in pull request #198 and merged at commit `182b7d3`. Error messages were added to the deficient `require` statements in `BasePaymaster.sol` and `DepositPaymaster.sol`, and the `require` statement in `SimpleAccount.sol` was eliminated as part of a code change.*

Missing recommended function [samples]

The EIP states that an aggregated account should support the `getAggregationInfo` function, and that this function should return the account's public key, and possibly other data. However, the `BLSAccount` contract does not contain a `getAggregationInfo` function. Consider renaming the `getBlsPublicKey` function to `getAggregationInfo`.

Update: *Resolved in pull request #199 and merged at commit `12d2ac0`. The EIP now uses the `getBlsPublicKey` function as an example.*

Uninitialized implementation contract [samples]

The `SimpleAccountFactory` creates a new implementation contract but does not initialize it. This means that anyone can initialize the implementation contract to become its owner.

The consequences depend on the version of OpenZeppelin contracts in use. The project requires release 4.2 and later, but release 4.8 is locked. The `onlyProxy` modifier was introduced in release 4.3.2 to protect the upgrade mechanism. Without this modifier, the owner is authorized to call the upgrade functions on the implementation contract directly, which lets them `selfdestruct` it.

With the locked version, the implementation owner can execute arbitrary calls from the implementation contract, but should not be able to interfere with the operation of the proxies.

Nevertheless, to reduce the attack surface, consider restricting the versions of OpenZeppelin contracts that are supported and disabling the initializer in the constructor of

Unrestrained revert reason [core]

The `EntryPoint` contract can emit a `FailedOp` error where the `reason` parameter provides additional context for troubleshooting purposes. However, there are two locations (line 375 and line 417) where an untrusted contract can provide the reason, potentially including misleading error codes. For example, the sender `validateUserOp` function might revert with `"AA90 invalid beneficiary"`, which might cause confusion during simulation.

Consider prefixing the externally provided revert reasons with a uniquely identifying error code.

Update: *Resolved in pull request #200 and merged at commit* `3d8f450`.

Unsafe ABI encoding

It is not an uncommon practice to use `abi.encodeWithSignature` or `abi.encodeWithSelector` to generate calldata for a low-level call. However, the first option is not safe from typographical errors, and the second option is not type-safe. The result is that both of these methods are error-prone and should be considered unsafe.

Within `EIP4337Manager.sol`, there are some occurrences of unsafe ABI encodings being used:

- On line 119
- On line 144

Consider replacing all occurrences of unsafe ABI encodings with `abi.encodeCall`, which checks whether the supplied values actually match the types expected by the called function, and also avoids typographical errors.

Note that a bug related to the use of string literals as inputs to `abi.encodeCall` was fixed in version 0.8.13, so developers should exercise caution when using this function with earlier versions of Solidity.

Notes & Additional Information

Declare `uint/int` as `uint256/int256` [core and samples]

Throughout the codebase, there are multiple instances of `int` and `uint` being used, as opposed to `int256` and `uint256`. In favor of explicitness, consider replacing all instances of `int` with `int256`, and `uint` with `uint256`.

Update: *Partially resolved in pull request #215 and merged at commit `998fa7d`. Most instances have been addressed but there are some `uint` types remaining.*

File relocation recommendations [samples]

To provide additional clarity regarding whether a given contract file contains core, sample, or test code, consider the following recommendations to move project files:

- Within the `samples` directory, `TestAggregatedAccount.sol`, `TestAggregatedAccountFactory.sol`, and `TestSignatureAggregator.sol` contain test contracts similar to those found in the `contracts/test` directory. Consider relocating these files to the `contracts/test` directory.
- The `bls` and `gnosis` directories contain sample account implementations, but do not reside in the `samples` directory. Consider moving these items to the `samples` directory.

Update: *Resolved in pull request #217 and merged at commit `f82cbbb`.*

IAccount inheritance anti-pattern

The `IAggregatorAccount` interface extends the base `IAccount` interface by adding the ability to expose a signature aggregator associated with the account. To add support for handling aggregated user operations, the `validateUserOp` function in `IAccount` now includes an `aggregator` address parameter. Accounts not associated with an aggregator must provide a null address for this parameter. This represents an anti-pattern where a base class is aware of features only relevant to a derived class.

account-specific extensions.

Update: *Resolved in pull request #216 and merged at commit* `1f505c5`.

Implicit size limit [core]

The `packSigTimeRange` function of the `BaseAccount` contract implicitly assumes the timestamps fit within 8 bytes. Consider enforcing this assumption by using `uint64` parameters.

Update: *Resolved in pull request #203 and merged at commit* `fa46d5b`.

Incomplete event history [samples]

The `BLSAccount` contract emits an event when the public key is changed, but not when it is initialized. To complete the event history, consider emitting the event on initialization as well.

Update: *Resolved in pull request #204 and merged at commit* `2600d7e`.

Lack of indexed parameter [core]

The `aggregator` parameter in the `SignatureAggregatorChanged` event is not indexed. Consider indexing the event parameter to avoid hindering the task of off-chain services searching and filtering for specific events.

Update: *Resolved in pull request #202 and merged at commit* `1633c06`.

Naming suggestions [core and samples]

To favor explicitness and readability, there are several locations in the contracts that may benefit from better naming. Our suggestions are:

- In `BaseAccount.sol`:
  - The `packSigTimeRange` function is internal but is not prefixed with "_". Consider renaming to `_packSigTimeRange`.
- In `BasePaymaster.sol`:

- Consider renaming all instances of `hashPublicKey` to `publicKeyHash` for consistency.

- In `EIP4337Manager.sol`:
  - Consider renaming the local variable `_msgSender` to `msgSender` for consistency.
- In `IAggregator.sol`:
  - Consider renaming the return value of the `aggregateSignatures` function from `aggregatesSignature` to `aggregatedSignature`.
- In `IEntryPoint.sol`:
  - The `ExecutionResult` error uses `validBefore` instead of `validUntil`. For consistency, consider changing the parameter name to `validUntil`.
  - The `ReturnInfo` struct's documentation for the `validAfter` parameter indicates it is inclusive. Consider renaming it to `validFrom` throughout the entire codebase.
  - In the `AggregatorStakeInfo` struct, consider renaming `actualAggregator` to `aggregator` (also in the comment here).
- In `SenderCreator.sol`:
  - In the `createSender` function, consider renaming the `initAddress` variable to `factory` to be consistent with the EntryPoint contract.
- In `SimpleAccount.sol`:
  - In the `addDeposit` function, consider renaming the `req` variable to `success`.
- In `StakeManager.sol`:
  - `internalIncrementDeposit` is an internal function that uses "internal" as its prefix instead of "_". Consider changing to `_incrementDeposit`.
  - The `getStakeInfo` function is internal but not prefixed with "_". Consider renaming the function to `_getStakeInfo`.
  - Consider renaming the `addr` parameter of `getStakeInfo` to `account`.
  - Consider removing the leading underscore from all instances of `_unstakeDelaySec` in `StakeManager` now that there is no longer a storage variable named `unstakeDelaySec`.

Update: *Resolved in pull request #221 and merged at commit* `7bd9909`.

Inconsistent ordering [core and samples]

Within the codebase, this recommendation is not followed in several places:

- In `BLSAccount.sol`: The `PublicKeyChanged` event is defined between two functions.
- In `BLSSignatureAggregator.sol`: Constant value `N` is defined between two functions.
- In `IEntryPoint.sol`: Starting at line 70, error and struct definitions are intermingled with function definitions.
- In `IPaymaster.sol`: The `PostOpMode` enum is defined after all functions.
- In `SimpleAccount.sol`:
  The `_entryPoint` variable, `SimpleAccountInitialized` event,
  and `onlyOwner` modifier are defined after several function definitions.

To improve the project's overall legibility, consider standardizing ordering throughout the codebase, as recommended by the Solidity Style Guide.

Update: *Partially resolved in pull request #211 and merged at commit `ca1b649`.*
In `IEntryPoint.sol`, the error definitions were relocated but several struct definitions remain defined in between functions.

Stake size inconsistency [core]

The `StakeManager` allows deposits up to the maximum `uint112` value, but the stake must be strictly less than the maximum `unit112` value. Consider using the same maximum in both cases for consistency.

Update: *Resolved in pull request #209 at commit `419b7b0`.*

TODO comments [core and samples]

The following instances of `TODO` comments were found in the codebase:

- Line 305 in EntryPoint.sol
- Line 52 in EIP4337Manager.sol
- Line 57 in TokenPaymaster.sol

and supporting details may be lost. Consider removing all instances of `TODO` comments and tracking them in the issues backlog instead. Alternatively, consider linking each inline `TODO` to the corresponding issues backlog entry.

Update: *Resolved in pull request #218 and merged at commit `80d5c89`. The first example is obsolete. The other two are not TODOs and were changed to "Note".*

Typographical errors [core and samples]

Consider addressing the following typographical errors:

- In `BaseAccount.sol`:
  - Line 70: "chain-id" should be "chain id".
  - Line 76: "The an account" should be "If an account".
- In `BLSAccount.sol`:
  - Line 9: "public-key" should be "public key" in this context.
  - Line 12: "a BLS public" should be "a BLS public key".
  - Line 19: "Mutable values slots" should be "Mutable value slots".
- In `BLSAccountFactory.sol`:
  - Line 11: "Based n" should be "Based on".
  - Line 27: "public-key" should be "public key" in this context.
- In `BLSHelper.sol`:
- Line 32: "(x2 y2, z2)" should be "(x2, y2, z2)".
- Line 137: "Doubles a points" should be "Doubles a point".
- In `BLSSignatureAggregator.sol`:
  - Line 34: "to short" should be "too short".
  - Line 89: "public-key" should be "public key" in this context; remove 1 space between "value" and "using".
  - Line 155: remove 1 space between "stake" and "or".
- In `DepositPaymaster.sol`:
  - Line 14: "deposit" should be "deposits".
- In `EIP4337Manager.sol`:
  - Line 106: "prevent mistaken replaceEIP4337Manager to disable" should be "prevents mistaken replaceEIP4337Manager from disabling".

- Line 80: “UserOperation” should be “UserOperations” or “user operations”.
- Line 180: “except that” should be “except for”.
- Line 180: Missing closing parenthesis.
- Line 522: “if it is was” should be “if it was”.
- Line 552: “A50” should be “AA50”.
- Line 560: “A51” should be “AA51”.

- In `IAccount.sol`:
  - Line 29: “The an account” should be “If an account”.
- In `IAggregatedAccount.sol`:
  - Line 9: “account, that support” should be “account that supports”.
  - Line 11: “valiate” should be “validate”.
- In `IAggregator.sol`:
  - Line 20: “return” should be “returns”.
  - Line 20: Sentence ends with a colon.
  - Line 23: Missing closing parenthesis.
- In `IEntryPoint.sol`:
  - Line 118: “factor” should be “factory”.
  - Line 129: “factor” should be “factory”.
- In `IPaymaster.sol`:
  - Line 13: “agree” should be “agrees”.
  - Line 24: “validation,)” should be “validation)”.
  - Line 48: “Now its” should be “Now it’s”.
- In `IStakeManager.sol`:
  - Line 22: Docstring copy-paste error from line 29.
  - Line 51: “allow” should be “allows”.
- In `SimpleAccount.sol`:
  - Line 65: “transaction” should be “transactions”.
- In `TestAggregatedAccount.sol`:
  - Line 18: “Mutable values slots” should be “Mutable value slots”.
- In `TestAggregatedAccountFactory.sol`:
  - Line 10: “Based n” should be “Based on”.
- In `TokenPaymaster.sol`:

- In `UserOperation.sol`:
  - Line 16: "field hold" should be "field holds".
  - Line 16: "paymaster-specific-data" should be "paymaster-specific data"; also remove quotes around this phrase.

Update: *Resolved in pull request #219 and merged at commit* `b4ce311`.

Unused imports [samples]

Throughout the codebase imports on the following lines are unused and could be removed:

- Import `console` of `BLSSignatureAggregator.sol`
- Import `EIP4337Manager` of `EIP4337Fallback.sol`
- Import `Exec` of `GnosisAccountFactory.sol`
- Import `IAggregator` of `IAggregatedAccount.sol`
- Import `UserOperation` of `IAggregatedAccount.sol`
- Import `Ownable` of `DepositPaymaster.sol`
- Import `BaseAccount` of `TestAggregatedAccount.sol`
- Import `SimpleAccount` of `TestSignatureAggregator.sol`
- Import `console` in `TestSignatureAggregator.sol`
- Import `SimpleAccount` of `TokenPaymaster.sol`

Consider removing unused imports to avoid confusion that could reduce the overall clarity and readability of the codebase.

Update: *Resolved in pull request #206 and merged at commit* `e019bbd`.

Unused interface [core]

The `ICreate2Deployer.sol` import was removed from `EntryPoint.sol` in pull request #144, but the file still exists in the `interfaces` directory. None of the contracts import this file.

Consider deleting the unused interface file.

Update: *Resolved in pull request #205 and merged at commit* `679ac11`.

e.g. `SimpleWallet` was renamed `SimpleAccount`. However, some “wallet” references remain in various comments:

- Line 13 of BLSAccountFactory.sol
- Line 9 of GnosisAccountFactory.sol
- Line 12 of TestAggregatedAccountFactory.sol
- Line 14 of VerifyingPaymaster.sol
- Line 16 of VerifyingPaymaster.sol

To avoid confusion, consider replacing these instances of “wallet” with “account”.

Update: *Resolved in pull request #210 and merged at commit* `d6a2db7`.

Conclusions

One high severity issue was found. Several changes were proposed to improve the code’s overall quality and reduce the attack surface.

Appendix

Monitoring Recommendations

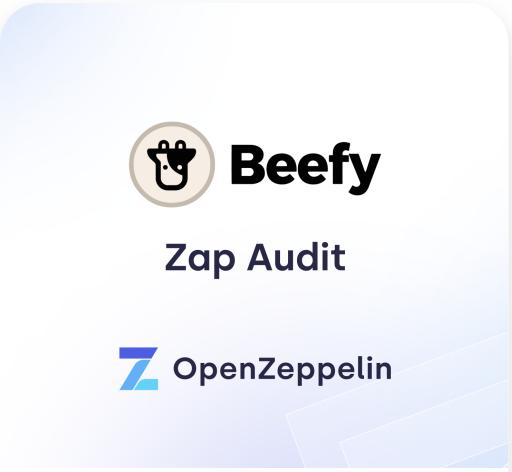
While audits help in identifying potential security risks, the Ethereum Foundation is encouraged to also incorporate automated monitoring of on-chain contract activity, and activity within the new mempool, into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment. In this case, it may also provide useful information about how the system is being used or misused. Consider monitoring the following items:

- User operations that have unusually high or low gas parameters may indicate a general misunderstanding of the system, or could identify unexpected economic opportunities in some kinds of transactions.
- Operations or paymasters that consistently fail validation in the mempool could indicate a misunderstanding of the system, or an attempted denial-of-service attack.

the specified restrictions.

- Operations where any of the participants have unusually low stake may provide useful insight into the risks that bundlers are willing to accept.

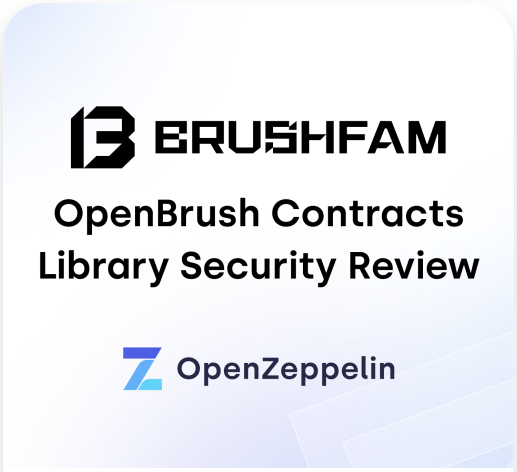
Related Posts



Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

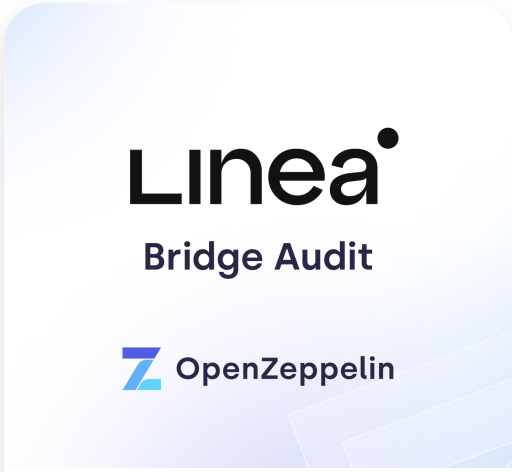
Security Audits



OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits



Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

Operation and Automation

Company

About us
Jobs
Blog

Contracts Library

Docs