Account Abstraction Security Review

Auditors

Gerard persoon, Lead Security Researcher

Kaden, Security Researcher

0xRajeev, Lead Security Researcher

Optimum, Lead Security Researcher

Report prepared by: Lucas Goiriz

March 25, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

The Ethereum Foundation is a non-profit organization that supports the Ethereum ecosystem that funds protocol development, grow the ecosystem and advocate for Ethereum.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Account Abstraction according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

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

3.1 Impact

- 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.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 16 days in total, Ethereum Foundation engaged with Spearbit to review the account-abstraction and ERCs (in particular `erc-4337.md` and `erc-7562.md`) repositories. In this period of time a total of **96** issues were found.

Summary

Project Name	Ethereum Foundation
Repository	account-abstraction, ERCs
Commit	ed8a5c79, 0b018104, 62a7f8b3
Type of Project	Infrastructure, Account Abstraction
Audit Timeline	Feb 18th to Mar 6th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	15	15	0
Gas Optimizations	6	5	1
Informational	74	66	8
Total	**96**	**87**	**9**

The Cantina Managed team reviewed Ethereum Foundation's account-abstraction, ERCS/erc-4337.md and ERCS/erc-7562.md on the commit hashes 57f9a8d7, 25f2b667 and 609c02dc respectively and concluded that all issues were addressed, and no new vulnerabilities were identified.

5 Findings

5.1 Medium Risk

5.1.1 `BLOBHASH` and `BLOBBASEFEE` opcodes allow for a mass invalidation attack

Severity: Medium Risk

Context: erc-7562.md#L158

Summary: The `BLOBHASH` and `BLOBBASEFEE` opcodes can be used to access environment information and execute a mass invalidation attack.

Finding Description: In the `ERC-7562` spec, we specify a list of blocked opcodes to prevent access of information of the environment. This is done to prevent an attacker from determining whether the validation logic of a `UserOperation` is being executed in an offchain simulation or onchain. We aim to avoid this to prevent mass invalidation attacks, which is an attack vector whereby many `UserOperation`s can be provided which appear to be valid, causing bundlers to perform necessary computation, or even pay gas, without these costs ever being repaid.

The `BLOBHASH` opcode allows us to access environment information, and thus determine if current execution is onchain, via determining whether a blob versioned hash has yet been provided for a given index. In case a blob versioned hash has not yet been provided for a given index, `BLOBHASH` will output zero to the stack, else it will always output a non-zero value. As such, knowing that a hash will be provided for a given index, we can ascertain whether we're onchain based on whether the output of a given index is non-zero.

The `BLOBBASEFEE` opcode allows us to access environment information based on whether the output value has changed or not. Each block, the `BLOBBASEFEE` returns an updated algorithmic value based on demand for blob space. As such, we can ascertain whether we're onchain based on whether the `BLOBBASEFEE` returns a new value. While it's possible for the `BLOBBASEFEE` to remain the same between blocks, it can be precomputed to determine whether this will occur.

Impact Explanation: Executing this variant of mass invalidation attack results in attack `UserOperation`s being indistinguishable from valid `UserOperation`s, causing them to be executed onchain en masse. Since we execute the operations onchain and fail validation, the bundler is required to pay the gas costs. Additionally, valid `UserOperation`'s will be prevented from being executed.

Recommendation: Add `BLOBHASH` and `BLOBBASEFEE` to the list of blocked opcodes in `ERC-7562`.

Ethereum Foundation: Fixed in PR 901.

Spearbit: Fixed as recommended both in the ERC-7562 documentation PR 901 and bundler implementation PR 247.

5.2 Low Risk

5.2.1 Wrong index passed to `_validateAccountAndPaymasterValidationData()` may lead to misidentification of offending operations

Severity: Low Risk

Context: EntryPoint.sol#L258-L263, EntryPoint.sol#L602-L629

Description: The call to the function `_validateAccountAndPaymasterValidationData()`, in `handleAggregatedOps()`, passes the wrong value for `opIndex`. Unlike `handleOps()` which calls `_validateAccountAndPaymasterValidationData()` within a single `for` loop, `handleAggregatedOps()` calls this function within the inner `for` loop, which uses `i` as its loop index. While the outer `for` loop iterates over the different aggregators, the inner loop iterates over all the user operations for each aggregator.

Passing the inner loop loop index `i` for `opIndex` parameter of `_validateAccountAndPaymasterValidationData()` effectively passes a reference to an incorrect user operation. This index is used to report the failing user operation in scenarios of account/paymaster errors related to signatures or out of time range. The impact of this wrong index is that it may lead to misidentification of offending operations during offchain simulation by bundlers. This may incorrectly reduce reputations of entities in those legitimate operations.

The debug-by-FailedOp technique is supposedly being deprecated in favor or a tracing-based approach.

Recommendation: Consider changing the code to:

```
- _validateAccountAndPaymasterValidationData(i, ... );
+ _validateAccountAndPaymasterValidationData(opIndex, ... );
```

Ethereum Foundation: Fixed in PR 547.

Spearbit: PR 547 refactors `handleAggregatedOps()` and `handleOps()` to introduce `_iterateValidationPhase()`, which correctly accounts for `opIndex`.

5.2.2 `outOfTimeRange` incorrectly considers current `block.timestamp` as within range for `validAfter`

Severity: Low Risk

Context: EntryPoint.sol#L645

Description: `outOfTimeRange` should return true if current time is outside the time range of `validationData` in `_getValidationData()`. However, in the edge-case where `block.timestamp == data.validAfter`, it incorrectly considers current `block.timestamp` as within range for `validAfter`. In such cases, the account or paymaster expected invariant on their set `validAfter` field for the user operation is broken.

Recommendation: Consider changing the code in `_getValidationData()` to:

```
- outOfTimeRange = block.timestamp > data.validUntil || block.timestamp < data.validAfter;
+ outOfTimeRange = block.timestamp > data.validUntil || block.timestamp <= data.validAfter;
```

Ethereum Foundation: Fixed in PR 545.

Spearbit: Fixed as recommended.

5.2.3 ERC-4337 and ERC-7562 descriptions of Validation phases and methodology are unclear

Severity: Low Risk

Context: erc-4337.md#L2

Description: The descriptions/applications of Validation phases and methodology is difficult to understand in ERC-4337 and ERC-7562. There are conflicting references to the number of phases, what rules are applied in each of them, who performs those validations, how/when are they performed (offchain/onchain), the separation of validation and execution phases, the different techniques used, the rationale behind these approaches and the resulting consequences of failed validations.

Given that bundler implementations are expected to implement these accurately to avoid mass-invalidation attacks, any misinterpretation of these aspects can result in DoS attacks.

Recommendation: Consider:

1. Defining Validation and distinguishing its phases.
2. Using the "frame" terminology from RIP-7560.
3. Referencing the Validation phases accurately.
4. Mapping rules to the Validation phases.
5. Mapping entities enforcing the Validation rules.
6. Distinguish offchain versus onchain enforcement.
7. Consequences of failed validations.
8. Clarifying the roles of different techniques applied:
    - Use of transaction tracing.

- Use of contract `EntryPointSimulations` for gas estimations.
- Use of `delegateAndRevert()`.

Ethereum Foundation: Fixed in PR 938 and others.

Spearbit: Improved as recommended.

5.2.4 `BasePaymaster` inherits from `Ownable` which is risky

Severity: Low Risk

Context: BasePaymaster.sol#L16

Description: `BasePaymaster` inherits from OpenZeppelin `Ownable`, which allows single-step transfer of ownership and therefore is susceptible to mistakes.

Recommendation: Consider using OpenZeppelin `Ownable2Step`, which as stated:

> This extension of the {Ownable} contract includes a two-step mechanism to transfer ownership, where the new owner must call {acceptOwnership} in order to replace the old one. This can help prevent common mistakes, such as transfers of ownership to incorrect accounts, or to contracts that are unable to interact with the permission system.

Ethereum Foundation: Fixed in PR 548.

Spearbit: Fixed as recommended.

5.2.5 Missing ERC-7562 rule for aggregator may hurt its reputation when other entities are at fault

Severity: Low Risk

Context: erc-7562.md#L251-L257

Description: ERC-7562 defines a set of "Entity-specific Rules" that include `EREP-015` which defines that:

> A `paymaster` should not have its opsSeen incremented on failure of factory or account.

However, a similar rule is missing for aggregators. Bundlers not implementing such a rule may unfairly penalize the aggregator reputation even when other entities of account/paymaster/factory maybe at fault.

Recommendation: Consider introducing a rule similar to `EREP-015` that is applicable to aggregators.

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended by adding "*[EREP-016] An aggregator should not have its opsSeen incremented on failure of factory, account or paymaster*" to ERC-7562.

5.2.6 ERC-7562 Opcode Rules missing `0xa` and `0xb-0x11` precompiles

Severity: Low Risk

Context: erc-7562.md#L193-L196

Description: ERC-7562 Opcode Rule `OP-062` on Precompiles defines:

1. Only allow known accepted precompiles on the network, that do not access anything in the blockchain state or environment.
2. The core precompiles `0x1` ... `0x9`.
3. The RIP-7212 `secp256r1` precompile, on networks that accepted it.

However, (2) misses the precompile `0xa` on "Point evaluation" added in Dencun, and the upcoming `0xb-0x11` on "BLS12-381 curve operations" being added now in Pectra. Missing these core precompiles will lead to `UserOperations` not being able to use these unlisted precompiles, such as for BLS, which will be unexpected.

Recommendation: Consider extending (2) until "...`0x11`".

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended.

5.2.7 ERC-4337 has several mandatory rules that are incorrectly described as recommendations

Severity: Low Risk

Context: erc-4337.md#L80, erc-4337.md#L184, erc-4337.md#L231, erc-4337.md#L369, erc-4337.md#L389

Description: ERC-4337 has several places, where `SHOULD` is used (sometimes `should`) which suggests a recommendation. However `MUST` is more appropriate because it is considered as a mandatory rule by the ERC. Otherwise, bundlers may choose to disregard the recommendation which will lead to skipped rule validations.

Recommendation: Consider reviewing all usages of `should` to evaluate if they must be replaced by `MUST` so that the standard EIP/IETF terminology rfc2119 for MUST/SHOULD is followed.

Ethereum Foundation: Solved in PR 936.

Spearbit: Fixed as recommended.

5.2.8 Missing `receive` function in ERC-4337

Severity: Low Risk

Context: StakeManager.sol#L42-L44

Description: In the `EntryPoint` contract, we inherit `StakeManager`, which includes a `receive` function to handle incoming deposits:

```
receive() external payable {
    depositTo(msg.sender);
}
```

We also implicitly call this function from the `BaseAccount` contract to handle pre-funding logic:

```
function _payPrefund(uint256 missingAccountFunds) internal virtual {
    if (missingAccountFunds != 0) {
        (bool success,) = payable(msg.sender).call{
                value: missingAccountFunds
            }("");
        (success);
        //ignore failure (its EntryPoint's job to verify, not account.)
    }
}
```

However, we do not specify in ERC-4337 that the `EntryPoint` contract must contain a `receive` function. As such, an `EntryPoint` implementation may be spec compliant, yet be incompatible with a contract inheriting `BaseAccount`.

Recommendation: Include the `receive` function in the defined `EntryPoint` interface in ERC-4337.

Ethereum Foundation: Fixed in PR 912.

Spearbit: Fixed as recommended.

5.2.9 ERC-4337 missing an Account Contract recommendation for invalid signatures will lead to inaccurate gas estimation

Severity: Low Risk

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

Description: ERC-4337 describes the below mandatory and recommended actions for the Account Contract interface:

> The account:
>
> - MUST validate the caller is a trusted EntryPoint.
> - MUST validate that the signature is a valid signature of the `userOpHash`, and. SHOULD return SIG_VALIDATION_FAILED (and not revert) on signature mismatch. Any other error MUST revert.
> - MUST pay the entryPoint (caller) at least the "missingAccountFunds" (which might be zero, in case the current account's deposit is high enough).
> - The account MAY pay more than this minimum, to cover future transactions (it can always issue `withdrawTo` to retrieve it).
> - The return value MUST be packed of `authorizer`, `validUntil` and `validAfter` timestamps.
>   - authorizer - 0 for valid signature, 1 to mark signature failure. Otherwise, an address of an authorizer contract, as defined in ERC-7766.
>   - `validUntil` is 6-byte timestamp value, or zero for "infinite". The UserOp is valid only up to this time.
>   - `validAfter` is 6-byte timestamp. The UserOp is valid only after this time.
>
> The account MAY implement the interface `IAccountExecute`.

However, it misses to include a recommended action to not return early in the `SIG_VALIDATION_FAILED` scenario. Account Contract signature validators should not return early for invalid signatures because bundler simulation is performed with mock signatures and doing so provides inaccurate gas estimates, which may later lead to OOG reverts.

Recommendation: Consider adding a suitable recommended behavior to the above list for Account Contract to do the normal flow as long as possible on invalid signature path to allow for an accurate gas estimation.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended by adding: "*SHOULD not return early when returning `SIG_VALIDATION_FAILED`. Instead, it SHOULD complete the normal flow to enable performing a gas estimation for the validation function.*" to ERC-4337.

5.2.10 Bundlers must drop `UserOperation`s which are not `validUntil` the next block

Severity: Low Risk

Context: erc-4337.md#L368

Description: In ERC-4337, we expect `validationData` to be returned from `validateUserOp` and `validatePaymasterUserOp`. Part of this `validationData` is the `validAfter` and `validUntil` values, defining the time range in which the `UserOperation` may be executed:

- `validUntil` is 6-byte timestamp value, or zero for "infinite". The `UserOperation` is valid only up to this time.
- `validAfter` is 6-byte timestamp. The `UserOperation` is valid only after this time.

One of the invariants of the system is that it must not be possible for the validation of a `UserOperation` to succeed in one of the early simulation steps but fail in a later one, or onchain, else a mass invalidation attack can be executed. If a `UserOperation` is submitted, with a `validUntil` value that is after the current timestamp, but before the next block timestamp, it could pass at least the initial validation step.

It's important to note that ERC-4337 notes that:

> A node MAY drop a `UserOperation` if it expires too soon (e.g. wouldn't make it to the next block) by either the `account` or `paymaster`.

This is a good recommendation, but "MAY" should be replaced with "MUST", specifically for the case that it wouldn't make it to the next block, otherwise a mass invalidation attack would be possible on bundlers which do not enforce this.

Recommendation: Update ERC-4337 to note that bundlers MUST drop a `UserOperation` if it expires before the next block.

Ethereum Foundation: Fixed in PR 957.

Spearbit: Fixed as recommended.

5.2.11 Unenforced `paymasterVerificationGasLimit` maximum

Severity: Low Risk

Context: erc-4337.md#L298

Description: In ERC-4337, we enforce a `MAX_VERIFICATION_GAS` value which the provided `verificationGasLimit` must not exceed for the `UserOperation` to be included:

> When a bundler receives a `UserOperation`, it must first run some basic sanity checks, namely that: [...]
> The `verificationGasLimit` is sufficiently low (<= `MAX_VERIFICATION_GAS`)[...]

Enforcing this maximum is important because any unused verification gas will be refunded to the `sender`, but the bundler must assume that the full amount will be consumed, taking up space in the bundle which could otherwise be used to include other `UserOperation`s.

The problem lies in the fact that the specification does not also enforce a maximum amount of gas for the `paymasterVerificationGasLimit`. As such, it's possible to provide a `UserOperation` with a very large `paymasterVerificationGasLimit`. Under the current specification, this `UserOperation` would be accepted in the bundle and would prevent other `UserOperation`s from being included as well due to the limited block space which appears to be consumed.

Recommendation: As enforced in the case of the `verificationGasLimit`, enforce a maximum for the `paymasterVerificationGasLimit`, beyond which the bundler is expected to drop the `UserOperation`.

Ethereum Foundation: Fixed in PR 941.

Spearbit: Fixed by updating ERC-4337 to enforce that the `paymasterVerificationGasLimit` is also less than the `MAX_VERIFICATION_GAS`.

5.2.12 ERCs miss recommending a limit check to the context size returned by `validatePaymasterUserOp`

Severity: Low Risk

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

Description: ERC-4337 says:

> If the paymaster's validatePaymasterUserOp returns a "context", then `handleOps` must call `postOp` on the paymaster after making the main execution call.

However, it does not mandate/recommend bundlers to check the size of context returned by `validatePaymasterUserOp` and enforcing a limit on its length. Otherwise, malicious paymasters can cause `EntryPoint` to consume a lot of memory in processing their returned contexts and potentially force OOG exceptions while copying them into memory.

Recommendation: Consider recommending a limit check to the context size returned by `validatePaymasterUserOp` either in ERC-7562 or ERC-4337.

Ethereum Foundation: Fixed in PR 948.

Spearbit: Fixed as recommended by adding "*The `context` byte array size MUST be below `CONTEXT_MAX_LEN` (2048) bytes, otherwise the `UserOperation` is considered invalid.*" to ERC-4337.

5.2.13 ERC-7562 is missing an Opcode Rule to allow calling `incrementNonce()` from the sender

Severity: Low Risk

Context: erc-7562.md#L185-L190

Description: ERC-7562 in its Opcode Rules has the below set of rules specifying allowed accesses to `EntryPoint`:

```
Allowed access to the `EntryPoint` address:
    * [OP-051] May call `EXTCODESIZE ISZERO`\
      This pattern is used to check destination has a code before the `depositTo` function is called.
    * [OP-052] May call `depositTo(sender)` with any value from either the `sender` or `factory`.
    * [OP-053] May call the fallback function from the `sender` with any value.
    * [OP-054] Any other access to the `EntryPoint` is forbidden.
```

However, this is missing a reference to `incrementNonce()`, which is inherited by `EntryPoint` from `NonceManager`:

```
function incrementNonce(uint192 key) public override {
    nonceSequenceNumber[msg.sender][key]++;
}
```

This may prevent valid nonce increment calls by UserOperations and cause them to be invalidated.

Recommendation: Consider adding a rule to explicitly allow a call to `incrementNonce()`.

Ethereum Foundation: Fixed in PR 565.

> `incrementNonce()` is left in to avoid modifications to the public ABI of the EntryPoint. Removed the misleading comment.

Spearbit: Fixed by removing the misleading comment.

5.2.14 Bundler must pay for dynamic gas cost which is assumed to be static

Severity: Low Risk

Context: EntryPoint.sol#L108-L118

Description: In `EntryPoint._executeUserOp`, we track the gas usage associated with preparing and executing the `innerHandleOp` call, but we only actually use this tracked amount if `innerHandleOp` reverts, else we will just use the gas consumed within the call, ignoring any gas used in `_executeUserOp`.

We expect this to not be an issue under the assumption that these gas costs are all static and are covered under `preVerificationGas`, which has a minimum amount enforced to ensure that enough is paid to cover all the static gas costs. However, since we have an arbitrarily sized `userOp` and `opInfo`, both encoding and executing the call will vary accordingly. This is exacerbated in the case that we already have a significantly large memory, via quadratic memory expansion costs, which is likely considering the fact that during verification, all `userOps` have been copied to memory.

```
if (methodSig == IAccountExecute.executeUserOp.selector) {
    bytes memory executeUserOp = abi.encodeCall(IAccountExecute.executeUserOp, (userOp,
    ↪  opInfo.userOpHash));
    innerCall = abi.encodeCall(this.innerHandleOp, (executeUserOp, opInfo, context));
} else
{
    innerCall = abi.encodeCall(this.innerHandleOp, (callData, opInfo, context));
}
assembly ("memory-safe") {
    success := call(gas(), address(), 0, add(innerCall, 0x20), mload(innerCall), 0, 32)
    collected := mload(0)
}
```

As such, we can't safely depend on the `preVerificationGas` provided to be sufficient to cover the dynamic costs included here. This is especially true considering the fact that the bundler cannot predict whether the call will revert or not. As a result, the bundler will have to pay any excess gas incurred, potentially making the entire bundle unprofitable.

Recommendation: To resolve this, we must either redesign the gas accounting logic used here or impose limits to the size of the `userOp`, increasing the `PRE_VERIFICATION_OVERHEAD_GAS` constant to account for the maximum cost incurred via this logic. The former may be a better option, especially since it is aligned with the finding, "`innerHandleOp()` could be simplified".

Ethereum Foundation: Fixed in PR 955 by adding documentation as to how to safely estimate `preVerificationGas`.

Spearbit: Fix verified.

5.2.15 `initEip7702Sender()` not using `verificationGasLimit` causes validation gas accounting to be inaccurate for EIP-7702 operations

Severity: Low Risk

Context: EntryPoint.sol#L444-L455

Description: In `_createSenderIfNeeded()`, while the call to `senderCreator().createSender()` uses `opInfo.mUserOp.verificationGasLimit`, the call to `senderCreator().initEip7702Sender()` does not do so as shown below:

```
if ( Eip7702Support._isEip7702InitCode(initCode) ) {
    if (initCode.length>20 ) {
        //already validated it is an EIP-7702 delegate (and hence, already has code)
        senderCreator().initEip7702Sender(sender, initCode[20:]);
    }
    return;
}
if (sender.code.length != 0)
    revert FailedOp(opIndex, "AA10 sender already constructed");
address sender1 = senderCreator().createSender{
    gas: opInfo.mUserOp.verificationGasLimit
}(initCode);
```

This omission causes validation gas accounting to be inaccurate for EIP-7702 UserOperations in the bundle.

Recommendation: Consider using `gas: opInfo.mUserOp.verificationGasLimit` in the call to `senderCreator().initEip7702Sender(sender, initCode[20:])`.

Ethereum Foundation: Fixed in PR 564.

> While we fixed it for completeness, the outcome difference is much less subtle:
>
> the entire validation is covered by verificationGasLimit (here), so neither init, norEip7702, not senderCreator, nor even validateUserOp can use validationGasLimit entirely, since it has to cover them all, including the cost of preparing the calldata for the calls, and nonce update.
>
> The real difference if we remove these per-method gas-limits would be reverts on different errors.

Spearbit: Fixed as recommended.

5.3 Gas Optimization

5.3.1 Public functions could be external

Severity: Gas Optimization

Context: EntryPoint.sol#L181-L184, EntryPoint.sol#L215-L218, EntryPoint.sol#L369, EntryPoint.sol#L473, NonceManager.sol#L26, StakeManager.sol#L19-L21, StakeManager.sol#L73

Description: Several functions are currently defined as `public` but are never called from within the contract.

Recommendation: Consider changing them to `external` to save some gas.

Ethereum Foundation: Fixed in PR 545 and PR 558.

Spearbit: Fixed as recommended.

5.3.2 Initialize with 0 not really necessary

Severity: Gas Optimization

Context: EntryPoint.sol#L189

Description: In several places a variable is initialized with 0, especially in for loops. This is not really necessary.

Recommendation: Consider removing the `= 0`. Note: this can reduce readability and the gas savings are minimal.

Ethereum Foundation: Acknowledged.

Spearbit: Acknowledged.

5.3.3 Function `handleAggregatedOps()` doesn't use `unchecked`

Severity: Gas Optimization

Context: EntryPoint.sol#L215

Description: Function `handleAggregatedOps()` doesn't use `unchecked`, while several other functions like `handleOps()` do use it.

Recommendation: Consider adding an `unchecked` block.

Ethereum Foundation: Fixed in PR 565.

Spearbit: Fixed as recommended.

5.3.4 A calculation in `_getUnusedGasPenalty()` can be simplified

Severity: Gas Optimization

Context: EntryPoint.sol#L861

Description: The calculation:

```
if (gasLimit <= gasUsed || gasLimit - gasUsed <= PENALTY_GAS_THRESHOLD)
```

Is the same as:

```
if (gasLimit <= gasUsed || gasLimit <= gasUsed + PENALTY_GAS_THRESHOLD)
```

Which can be simplied to:

```
if (gasLimit <= gasUsed + PENALTY_GAS_THRESHOLD)
```

Recommendation: Consider changing the code to:

```
- if (gasLimit <= gasUsed || gasLimit - gasUsed <= PENALTY_GAS_THRESHOLD)
+ if (gasLimit <= gasUsed + PENALTY_GAS_THRESHOLD)
```

Ethereum Foundation: Fixed in PR 545.

Spearbit: Fixed as recommended.

5.3.5 Function `withdrawTo()` can be optimized

Severity: Gas Optimization

Context: StakeManager.sol#L134-L145

Description: In the function `withdrawTo()`, `info.deposit` is retrieved twice. This can be optimized.

Recommendation: Consider storing the value of `info.deposit` in a temporary variable.

Ethereum Foundation: Fixed in PR 550.

Spearbit: Fixed as recommended.

5.3.6 Array length can be cached

Severity: Gas Optimization

Context: BaseAccount.sol#L65-L69

Description: In function `executeBatch()`, the array length is evaluated every cycle of the for loop, which is not gas efficient.

Recommendation: Consider caching `calls.length`.

Ethereum Foundation: Fixed in PR 550.

Spearbit: Fixed as recommended.

5.4 Informational

5.4.1 Parameters may need to be updated before release

Severity: Informational

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

Description: Some parameters need an update before the release of version 0.8:

- `package.json` still contains: `"version": "0.7.0"`.
- Different Solidity versions are used in the pragmas:
  - `pragma solidity >=0.7.5;`
  - `pragma solidity ^0.8;`
  - `pragma solidity ^0.8.23;`
- Several features require newer solidity versions:
  - `ReentrancyGuardTransient` uses `pragma solidity ^0.8.24;`
- Custom error support in `require` uses at least version `0.8.26`.

Recommendation: Before the release of version "V0.8", consider updating the following:

- `package.json`.
- Solidity versions used.

Ethereum Foundation: Addressed in PR 568.

Spearbit: Verified.

5.4.2 Rationale of using `getFreePtr()` and `restoreFreePtr()` is not obvious

Severity: Informational

Context: EntryPoint.sol#L98, EntryPoint.sol#L119, EntryPoint.sol#L526, EntryPoint.sol#L539, EntryPoint.sol#L845-L857

Description: The functions `getFreePtr()` and `restoreFreePtr()` are used to avoid unnecesary memory expansion: any memory allocation, such as the `abi.encodeCall()`, is never freed by solidity. The memory is left unused, and with a large bundle it could accumulate and cost more to execute the last `UserOps` in the bundle. However this might not be obvious to the readers of the code.

Recommendation: Consider adding comments to explain the use of `getFreePtr()` and `restoreFreePtr()`.

Ethereum Foundation: Addressed in PR 545.

Spearbit: Verified.

5.4.3 Layout, typos and outdated comments reduce readability

Severity: Informational

Context: BasePaymaster.sol#L28-L29, EntryPoint.sol#L131-L132, Helpers.sol#L30, Helpers.sol#L39, IAccount.sol#L26-L27, IPaymaster.sol#L32, PackedUserOperation.sol#L8, erc-4337.md#L450

Description: The codebase has minor issues with layout, typos and outdated comments, which reduce readability.

Recommendation: Consider making the following changes:

```diff
- //can only be caused by bundler
+ // can only be caused by bundler
```

```diff
- //sanity check: make sure this EntryPoint was compiled against the same
+ // sanity check: make sure this EntryPoint was compiled against the same
```

```diff
- * @param initCode             - If set, the account contract will be created by this constructor/
+ * @param initCode             - If set, the account contract will be created by this constructor
```

```diff
- * @param validaUntil - This UserOp is valid only up to this timestamp.
+ * @param validUntil - This UserOp is valid only up to this timestamp.
```

```diff
- * Extract sigFailed, validAfter, validUntil.
+ * Extract aggregator, validAfter, validUntil.
```

```diff
- sigAuthorizer
+ aggregatorOrSigFail
```

```diff
- "authorizer"
+ "aggregator"
```

- **Typos in ERC-4337:**

```diff
- 2. UserOperations that are valid when checked independently, by fail when bundled together to
↪  be put on-chain.
+ 2. UserOperations that are valid when checked independently, but fail when bundled together to
↪  be put on-chain.
```

Ethereum Foundation: Fixed in the following PRs: PR 545, PR 550, PR 558 and PR 569.

ERC4337 is fixed in PR 927.

Spearbit: Verified.

5.4.4 Missing Natspec for functions reduces readability

Severity: Informational

Context: Eip7702Support.sol#L18-L19, Eip7702Support.sol#L31-L32, Eip7702Support.sol#L47-L50, EntryPoint.sol#L160, EntryPoint.sol#L172, EntryPoint.sol#L413-L417, EntryPoint.sol#L479-L487, EntryPoint.sol#L522, EntryPoint.sol#L550-L560, EntryPoint.sol#L650-L662, EntryPoint.sol#L722-L735, Helpers.sol#L39-L45, Helpers.sol#L56-L61, Helpers.sol#L70-L78, Helpers.sol#L86-L89, Helpers.sol#L101-L104, NonceManager.sol#L22-L26, SenderCreator.sol#L53-L58

Description: Most functions have some description and document the parameters and return values. However, this is missing in some places .

Recommendation: Consider adding function descriptions and Natspec documentation of parameters and return values, either on the main contract, or via `@inheritdoc` on an interface. Already provided documentation for `emitUserOperationEvent()` and `emitPrefundTooLow()`:

> These methods are wrappers for the "emit" call, so that an overriding contract (e.g. EntryPointSimulations) to override, and maybe use the values in another way. In many places we added "virtual", to allow such "sub-classing" the contract. Eventually, we're not using most of this "override placeholders" in Simulations (and use debug tracing instead) The method name (and parameters) are those of the emitted event (with the only change that we pass the struct and decode inner fields inside the method).

Ethereum Foundation: Fixed in PR 550 and PR 569.

Spearbit: Fixed as recommended.

5.4.5 Function ordering within the contract deviates from Solidity style guide

Severity: Informational

Context: EntryPoint.sol#L181-L184, EntryPoint.sol#L215-L218

Description: In the `EntryPoint` contract, the `public/external` functions `handleOps()` and `handleAggregatedOps()` are in the middle of the contract. The Solidity style guide places `public/external` functions in the beginning, which improves readability.

Recommendation: Consider moving the `public/external` functions to the beginning of the contract.

Ethereum Foundation: Fixed in PR 574.

Spearbit: Verified.

5.4.6 Reason for `emit SignatureAggregatorChanged(address(0))` is not obvious

Severity: Informational

Context: EntryPoint.sol#L283

Description: The reason for `emit SignatureAggregatorChanged(address(0))` is:

> We wanted the sequence of events (emitted by the entrypoint) to be enough to know the status of entities.
>
> `handleAggregatedOps()` emit `SignatureAggregatorChanged` before each group of `UserOps` using that `aggregator`. But in case `handleOps()` is called just afterwards, we wanted to make sure the first `UserOperationEvent` of the `handleOps()` doesn't be mistakenly get associated with the last used aggregator.

However this is not obvious for anyone reading the code.

Recommendation: Consider adding a comment in the code explaining the reason.

Ethereum Foundation: Removed in PR 565. It seems to no longer be needed, since we added an event `BeforeExecution()` in `handleOps()` and `handleAggregatedOps()`.

Spearbit: Verified.

5.4.7 Validation of EIP-7702 delegate is not obvious

Severity: Informational

Context: EntryPoint.sol#L378, EntryPoint.sol#L446, EntryPoint.sol#L667

Description: A comment in function `_createSenderIfNeeded()` says `already validated it is an EIP-7702 delegate`, however, it's not obvious where this validation is done.

It is validated in `_validatePrepayment()` → `getUserOpHash()` → `_getEip7702InitCodeHashOverride()` → `_getEip7702Delegate()`.

It is also not obvious that `getUserOpHash()` retrieves the `EIP-7702 delegate` and adapts the hash based on that.

Recommendation: Consider adding a comment to the call of `getUserOpHash()` in `_validatePrepayment()`. Consider expanding the comment in function `_createSenderIfNeeded()` with a reference to the function `_getEip7702InitCodeHashOverride()`.

Ethereum Foundation: Fixed in PR 558.

Spearbit: Verified.

5.4.8 `initEip7702Sender()` can be called multiple times which is not obvious

Severity: Informational

Context: EntryPoint.sol#L444-L450

Description: `initEip7702Sender()` can be called multiple times which is not obvious.

Recommendation: Consider adding an explanation to clarify this aspect.

Ethereum Foundation: Fixed in PR 558.

Spearbit: Verified.

5.4.9 EIP-7702 information is limited in ERC-4337 and ERC-7562

Severity: Informational

Context: erc-4337.md#L2, erc-4337.md#L64, erc-4337.md#L228, erc-4337.md#L480-L482, erc-4337.md#L506, erc-7562.md#L2, erc-7562.md#L306

Description: Support for EIP-7702 will be soon added in the Pectra upgrade. However, the documentation about it in ERC-4337 and ERC-7562 is very limited. There are some updates in PR 882.

Recommendation: Consider adding the following to ERC-4337:

- EIP-7702 authorization tuple.
- The EIP-7702 authorization tuple must be valid.
- EIP-7702 authorization tuples are provided alongside the `UserOperation` struct, but they are not included in the `UserOperation` itself.
- First-time account creation.
- Add a description for 7702 accounts.
- Initializing delegated accounts:
- Called via `initEip7702Sender()`.
- `initEip7702Sender()` can be called multiple times to the smart wallet code should only allow it once.
- The init code must check `msg.sender==entryPoint.senderCreator`.

- `UserOperation`.
- Expand description of field `factoryData` with info about 7702.
- `bundle-seq.svg` flowchart.
- Add flow for 7702.
- Backwards Compatibility.
- This section should be updated because 7702 adds backwards compatibility.

Consider adding the following to ERC-7562:

- AUTH-010:
  - If multiple `UserOps` (of the same sender) have an `auth tuple` in the mempool, they all must have the same delegate.
  - If they have different nonces, the bundler should pick the right one.

Ethereum Foundation: Solved in PR 957 and PR 958.

Spearbit: Verified.

5.4.10 `getSenderAddress()` doesn't work for 7702 accounts

Severity: Informational

Context: EntryPoint.sol#L473-L476

Description: When 7702 init code is used, then `createSender()` will call the address `0x7702....0` and sender will be 0. In theory address `0x7702....0` can host a precompile on some chains. However, the function always ends with a revert so little harm can be done. Note: the function is meant to be used by wallets offchain and isn't used by the account abstraction contracts.

Recommendation: At least document that this function should not be used for `7702` based accounts. Preferably check for `INITCODE_EIP7702_MARKER` in the init code to give an appropriate error message.

Ethereum Foundation: Fixed in PR 558.

Spearbit: Fixed by adding recommended documentation.

5.4.11 Location of `opIndex` parameter in function parameters

Severity: Informational

Context: EntryPoint.sol#L487-L488, EntryPoint.sol#L524

Description: In most functions the parameter `opIndex` is the first parameter, however in `_callValidateUserOp()` it is the last parameter, which is inconsistent.

Recommendation: Consider moving `opIndex` to be the first parameter.

Ethereum Foundation: Fixed in PR 545.

Spearbit: Fixed as recommended.

5.4.12 Memory pointer manipulation is risky in `_callValidateUserOp()`

Severity: Informational

Context: EntryPoint.sol#L524-L547

Description: In the function `_callValidateUserOp()`, the variables `success` and `sender` are still used after the free memory pointer is reset. Luckily that is ok, but if `success` or `sender` would be a `memory` variable, it would not be ok.

Recommendation: Consider using the same pattern as `_executeUserOp()`:

```
bool success;
address sender=/*...*/;
{
    uint256 saveFreePtr = getFreePtr();
    // assign success & sender
    restoreFreePtr(saveFreePtr);
}
// use success & sender
```

Ethereum Foundation: Fixed in PR 545.

Spearbit: Verified.

5.4.13 Direct access to storage of `StakeManager` reduces readability and maintainability

Severity: Informational

Context: EntryPoint.sol#L487, EntryPoint.sol#L512-L517, EntryPoint.sol#L560, EntryPoint.sol#L570-L575

Description: The functions `_validateAccountPrepayment()` and `_validatePaymasterPrepayment()` directly access storage of `StakeManager`. This is more difficult to understand and maintain.

Recommendation: Consider adding a function `_decrementDeposit()` in `StakeManager` and using it in functions `_validateAccountPrepayment()` and `_validatePaymasterPrepayment()`.

Ethereum Foundation: Fixed in PR 565.

Spearbit: Verified.

5.4.14 The readability of `revert` statement could be increased

Severity: Informational

Context: EntryPoint.sol#L689-L691

Description: The readability of the `revert` statement could be increased. With the latest solidity versions, this could be the following, which might be easier to read:

Recommendation: Consider changing the code to:

```
- if (!_validateAndUpdateNonce(...)) { revert FailedOp(opIndex, "AA25 ..."); }
+ require(_validateAndUpdateNonce(...), FailedOp(opIndex, "AA25 ..."));
```

Ethereum Foundation: Fixed in PR 545.

Spearbit: Fixed as recommended.

5.4.15 `innerHandleOp()` could be simplified

Severity: Informational

Context: EntryPoint.sol#L87, EntryPoint.sol#L150-L155, EntryPoint.sol#L321, EntryPoint.sol#L365, EntryPoint.sol#L730, EntryPoint.sol#L757, EntryPoint.sol#L776

Description: The function `_postExecution()` is called twice and the second time it is called with `postOpReverted`. Inside the function `_postExecution()` most of the actions are skipped when its called the second time. This is a residue of previous versions of the entrypoint, but now is unnecessarily complex and error prone.

Recommendation: Consider simplifying the code. Here is pseudo code for execution:

```
function _executeUserOp()
    check there is sufficient gas
    success = call innerHandleOp
    do remaining gas accounting

function innerHandleOp()
    success1 = call executeUserOp
    success2 = call postOp with info about gas costs so far
    if success1==false or success2==false or prefund is insuffient: revert with info about gas usage
    return with info about gas usage
```

Ethereum Foundation: Acknowledged. Such a refactor would amount to a major change in the most sensitive parts of the EntryPoint contract very late in the development cycle for the "minor" v0.8 update. Such a change would lose the benefit of having most of our codebase audited multiple times.

Spearbit: Acknowledged.

5.4.16 No explicit check on length in `createSender()`

Severity: Informational

Context: SenderCreator.sol#L33

Description: In the function `createSender()`, there is no explicit check that `initCode.length >= 20`. If the `initCode` is too short, it will revert, but debugging might be more difficult.

Recommendation: Consider adding an explicit check and reverting with an error message.

Ethereum Foundation: Solved in PR 565 by adding the check in `EntryPoint::_createSenderIfNeeded()`.

Spearbit: Verified.

5.4.17 Function `getSender()` uses knowledge of struct ordering

Severity: Informational

Context: UserOperationLib.sol#L21-L28, PackedUserOperation.sol#L5-L6

Description: Function `getSender()` uses knowledge of the ordering of struct `PackedUserOperation`. If the struct is ever changed then the code doesn't work anymore.

Recommendation: Consider adding a comment in `PackedUserOperation.sol` to never change the first parameter.

Ethereum Foundation: Solved in PR 565 by removing `getSender()`.

Spearbit: Verified.

5.4.18 Function `unpackUints()` uses different algorithm than `unpackHigh128()` and `unpackLow128()`

Severity: Informational

Context: UserOperationLib.sol#L82-L96

Description: The function `unpackUints()` uses a different algorithm than `unpackHigh128()` and `unpackLow128()`. Both are correct, but using the same algorithm would be more consistent and easier to maintain.

Recommendation: Consider using the same algorithm.

Ethereum Foundation: Fixed in PR 558.

Spearbit: Verified.

5.4.19 Function `_packValidationData()` uses implicit knowledge of values instead of using declared constants

Severity: Informational

Context: Helpers.sol#L80

Description: Function `_packValidationData()` uses implicit knowledge of values 1 and 0. This makes the code more difficult to read.

Recommendation: Consider changing the code to use declared constants:

```
- (sigFailed ? 1 : 0)
+ (sigFailed ? SIG_VALIDATION_FAILED  : SIG_VALIDATION_SUCCESS )
```

Ethereum Foundation: Fixed in PR 550.

Spearbit: Verified.

5.4.20 `("memory-safe")` not always used

Severity: Informational

Context: EntryPoint.sol#L102-L106, UserOperationLib.sol#L26-L28

Description: Most uses of `assembly` also have `("memory-safe")`. However there are some exceptions where it is absent but could also be used.

Recommendation: Consider adding `("memory-safe")` everywhere where applicable.

Ethereum Foundation: Fixed in PR 565.

Spearbit: Verified.

5.4.21 Use of _ in function names is not consistent

Severity: Informational

Context: EntryPointSimulations.sol#L158-L162, EntryPointSimulations.sol#L210, EntryPoint.sol#L160, EntryPoint.sol#L172, EntryPoint.sol#L800-L802, EntryPoint.sol#L824-L826, EntryPoint.sol#L836-L838, EntryPoint.sol#L845, EntryPoint.sol#L853

Description: Some `external` functions start with _. The _ prefix typically means that it is an internal function. However, some internal functions do not use this convention.

Recommendation: Consider removing the _ from `external` functions and adding them to all internal functions.

Ethereum Foundation: Fixed in PR 565.

Spearbit: Verified.

5.4.22 The parameter name of `addStake()` is different in the interface

Severity: Informational

Context: StakeManager.sol#L73, IStakeManager.sol#L87

Description: The parameter name of `addStake()` is different in the interface than in the implementation contract.

Recommendation: Consider removing the _ in `IStakeManager.sol`.

Ethereum Foundation: Fixed in PR 550.

Spearbit: Fixed as recommended.

5.4.23 Function `executeBatch()` could be virtual

Severity: Informational

Context: BaseAccount.sol#L62

Description: Function `executeBatch()` of `BaseAccount.sol` could be `virtual` to allow overriding for "better" batching.

Recommendation: Consider marking `executeBatch()` as `virtual`.

Ethereum Foundation: Fixed in PR 550.

Spearbit: Fixed as recommended.

5.4.24 Inconsistent `validUntil` comments

Severity: Informational

Context: BaseAccount.sol#L112, Helpers.sol#L30, Helpers.sol#L71, IAccount.sol#L28, IPaymaster.sol#L34

Description: There are different comments used for `validUntil`.

Recommendation: For completeness and consistency, use the same comment everywhere.

Ethereum Foundation: Fixed in PR 558.

Spearbit: Verified.

5.4.25 Release version of ERC4337 `Entrypoint` cannot be retrieved

Severity: Informational

Context: EntryPoint.sol#L36

Description: There is no way to retrieve the version of the EntryPoint.

Recommendation: Consider adding a way to retrieve the version of the EntryPoint. This could be done via the `DOMAIN_VERSION`.

Ethereum Foundation: The version of the `entrypoint` should only be based on the actual `entrypoint` address. Changing `DOMAIN_VERSION` to "0.8" would not be a valid ERC-712 domain, it should be the "major version".

Spearbit: Acknowledged.

5.4.26 `_validateEntryPointInterface()` could do more checks

Severity: Informational

Context: BasePaymaster.sol#L30-L32

Description: The function `_validateEntryPointInterface()` is used to check the correct version of the `entry-point` is used. A newer version of the `entrypoint` might have the same `type(IEntryPoint).interfaceId`. Even if this is fully compatible a `paymaster` might want to make sure which version of the `entrypoint` is used.

Recommendation: Consider also checking the version of the `entrypoint`. See finding "Release version of ERC4337 `Entrypoint` cannot be retrieved" on how the version could be retrieved.

Ethereum Foundation: There is no authority to define the "version" of an `entrypoint`. The only security check is a published `entrypoint` address that is known and not a proxy in any way. The `supportsInterface` is a mere "nuance helper", e.g. to prevent deploying (and staking) a v0.6 `paymaster` with v0.7 `entrypoint`. As the interface was changed accessing an old entrypoint doesn't work.

Spearbit: Acknowledged.

5.4.27 Comment about `actualGasCost` is not accurate

Severity: Informational

Context: BasePaymaster.sol#L68-L82, EntryPoint.sol#L755-L760, IPaymaster.sol#L45-L57

Description: The comment about `actualGasCost` says `actual gas used so far`. However as can be seen in the call to `postOp()` from the function `_postExecution()`, it is already multiplied by the `gasPrice`. So the comment in not accurate.

Recommendation: Consider changing the comment in both `IPaymaster.sol` and `BasePaymaster.sol` so something like:

```
- * @param actualGasCost - Actual gas used so far (without this postOp call).
+ * @param actualGasCost - Actual gas used so far * gasPrice (without this postOp call).
```

Ethereum Foundation: Fixed in PR 558 and PR 545.

Spearbit: Fixed by replacing "`Actual gas..`" with "`Actual cost of gas..`".

5.4.28 No equivalent of `AccountDeployed` event emit for 7702 delegation

Severity: Informational

Context: EntryPoint.sol#L463-L468

Description: When an account is deployed via a factory, an `emit AccountDeployed` is done. However, if a 7702 delegation is done with an `EIP-7702 authorization tuple`, there is no `emit`. This makes it more difficult for indexers to trace the actions with accounts.

Recommendation: Consider adding an `emit` for `EIP-7702 authorization tuples`. This will require encoding extra information in the `struct PackedUserOperation`.

Ethereum Foundation: Unlike account creation, which we know the state change (created new account), with 7702 we don't know what it does, if it does anything at all. also, accounts can do initialization in `validateUserOp()`. No external entity (such as indexer) can rely on an event emitted by the `EntryPoint`. It must rely on account-specific event (that is, the initialization function should emit it, if at all).

Spearbit: Acknowledged.

5.4.29 Using the same `AA93` error code for two different checks may be misleading

Severity: Informational

Context: EntryPoint.sol#L402-L408

Description: While unique error codes are used everywhere else, `AA93` is used as the error code for two different but related checks:

```
require(
            paymasterAndData.length >= UserOperationLib.PAYMASTER_DATA_OFFSET,
            "AA93 invalid paymasterAndData"
);
```

```
require(paymaster != address(0), "AA93 invalid paymaster");
```

The error messages are also different. This could be misleading to anyone trying to map the error code to the nature of the error.

Recommendation: Consider adding a new error code to distinguish the two.

Ethereum Foundation: Fixed in PR 545.

Spearbit: Fixed as recommended by assigning a new error code AA98 for invalid paymaster.

5.4.30 ERC-7562 `UREP-011` is redundant

Severity: Informational

Context: erc-7562.md#L276

Description: ERC-7562 `UREP-011` defines that "A staked sender is only limited by the "Staked Entities Reputation Rules". However, this category of "Unstaked Entities Reputation Rules" applies only to unstaked entities. Therefore, this rule is either misclassified or redundant here.

Recommendation: Consider reclassifying `UREP-011` or removing it.

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended by removing it.

5.4.31 ERC-4337 `MAX_VERIFICATION_GAS` and `PRE_VERIFICATION_OVERHEAD_GAS` are undefined

Severity: Informational

Context: erc-4337.md#L298

Description: ERC-4337 notes that:

> The `verificationGasLimit` is sufficiently low (`<= MAX_VERIFICATION_GAS`) and the `preVerificationGas` is sufficiently high (enough to pay for the calldata gas cost of serializing the `UserOperation` plus `PRE_VERIFICATION_OVERHEAD_GAS`).

However, these threshold constants are not defined.

Recommendation: Consider defining these threshold constants and providing guidance on their typical values.

Ethereum Foundation: Fixed in PR 941.

Spearbit: Fixed as recommended.

5.4.32 ERC-7562 "Unstaked Paymasters Reputation Rules" section is mistitled

Severity: Informational

Context: erc-7562.md#L268

Description: ERC-7562 "Unstaked Paymasters Reputation Rules" section is not limited to paymasters but could apply to all unstaked entities including paymaster, accounts, aggregators and factories. Specifically, `UREP-010` applies to unstaked sender. However, the title suggests it is limited to paymasters.

Recommendation: Consider changing the section title to "Unstaked Entities Reputation Rules".

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended.

5.4.33 Comments in `IEntryPoint` only focus on `handleOps`

Severity: Informational

Context: IEntryPoint.sol#L95-L98, IEntryPoint.sol#L107-L110, IEntryPoint.sol#L119

Description: Several comments in `IEntryPoint` only focus on `handleOps`, while they are also relevant for `handleAggregatedOps`.

Recommendation: Consider changing the comments in the following way:

```diff
- handleOps
+ handleOps and handleAggregatedOps
```

Ethereum Foundation: Fixed in PR 558.

Spearbit: Fixed as recommended.

5.4.34 ERC-7562 `UREP-010` is missing the required condition on unstaked sender not being throttled/banned

Severity: Informational

Context: erc-7562.md#L275

Description: ERC-7562 `UREP-010` defines that "An unstaked sender is only allowed to have `SAME_SENDER_MEMPOOL_COUNT` `UserOperation`s in the mempool." However, unlike `UREP-020`, it is missing the required condition on unstaked sender not being throttled/banned.

Recommendation: Consider adding the condition on not being throttled/banned.

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended.

5.4.35 ERC-7562 classification of Network-wide and Local rules not being clear may lead to incorrect bundler implementations

Severity: Informational

Context: erc-7562.md#L49, erc-7562.md#L57

Description: ERC-7562 defines two types of validation rules: Network-wide rules and Local rules. However, the classification and explanation of these rules is not clear, which may lead to incorrect bundler implementations.

Recommendation: Consider clarifying in detail the classification and explanation of these rules.

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended.

5.4.36 Struct `AggregatorStakeInfo` is only used in `EntryPointSimulations.sol`

Severity: Informational

Context: IEntryPoint.sol#L197-L203

Description: Struct `AggregatorStakeInfo` is only used in `EntryPointSimulations.sol`.

Recommendation: Consider moving `struct AggregatorStakeInfo` from `IEntryPoint.sol` to `IEntryPointSimulations.sol`.

Ethereum Foundation: Fixed in PR 557.

Spearbit: Fixed as recommended.

5.4.37 ERC-7562 is missing two references to aggregator

Severity: Informational

Context: erc-7562.md#L225, erc-7562.md#L362

Description: ERC-7562 in its definition of rules for entities is missing two references to aggregator to be included with account, factory and paymaster:

1. "*This means that `Paymaster` and `Factory` contracts cannot practically be an "account" contract as well.*".
2. "*We want to be able to allow globally-used contracts (paymasters, factories) to use storage not associated with the account, but still prevent them from spamming the mempool.*".

This may be incorrectly inferred as such rules/explanations not being applicable to aggregators.

Recommendation: Consider adding references to aggregators wherever the rules/explanations apply to all entities.

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended.

5.4.38 ERC-4377 references to past events must be updated

Severity: Informational

Context: erc-4337.md#L29

Description: ERC-4377 in its Motivation section says:

> Ethereum consensus layer development is focusing on the merge and later on scalability-oriented features, and there may not be any opportunity for further protocol changes for a long time. Hence, to increase the chance of faster adoption, this proposal avoids Ethereum consensus changes.

The reference to focussing on Merge has to be updated given that it has already happened.

Recommendation: Consider updating references to past events and clarifying their context as per current status.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.39 ERC-4337 motivational use cases can be updated with better explanations

Severity: Informational

Context: erc-4337.md#L30-L33

Description: ERC-4337 motivational use cases can be updated with better explanations.

Recommendation: Consider:

1. Adding these from ERC-7562:
    - Abstracting the validation allows the contract to use different signature schemes, multisig configuration, custom recovery, and more.
    - Abstracting gas payments allows easy onboarding by 3rd party payments, paying with tokens, cross-chain gas payments.
    - Abstracting execution allows batch transactions.
2. Reference to EIP-3074 may be replaced with EIP-7702.
3. Replacing "Try to support other use cases" with "Enable support for other use cases".
4. Expanding on "Privacy-preserving applications".

Ethereum Foundation: Fixed in PR 943.

Spearbit: Fixed as recommended, except (4).

5.4.40 ERC-4337 missing definitions reduce readability

Severity: Informational

Context: erc-4337.md#L37-L53

Description: ERC-4337 Definitions section is missing definitions for deposit, factory, aggregator and mempools among others, which are referenced throughout the document. This reduces readability and leads to assumptions on their role and responsibilities.

Recommendation: Consider defining all the key terms used in ERC-4337, which are not part of historical terminology and are specifically being introduced by ERC-4337.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.41 ERC-4337's inconsistent use of "Sender" terminology reduces readability

Severity: Informational

Context: erc-4337.md#L43

Description: The "Sender" term is referenced in different ways throughout the document, including: Sender, Account Contract, account contract, account, smart contract wallets, SC, wallets etc... This inconsistent use of "Sender" terminology reduces readability.

Recommendation: Consider adopting a single term to reference this entity. Given this is about "Account Abstraction", "Account Contract" perhaps is the best fit.

Ethereum Foundation: Fixed in PR 938 and PR 942.

> The intended terminology is: Smart Contract Account - the contract being used, either source code or deployed instance. Account - shortened version if Smart Contract Account already appears in the sentence. sender - when referencing the entity as part and in the context of a UserOperation flow. The following terms are not intended and must not be used: wallet, SC, SCA, account, account contract etc...

Spearbit: Fixed as above.

5.4.42 ERC-4337 definition of `accountGasLimits` uses incorrect field names

Severity: Informational

Context: erc-4337.md#L92

Description: ERC-4337's Entrypoint definition defines `accountGasLimits` field of the Packed UserOperation struct as the concatenation of `verificationGas` and `callGas`, which is incorrect because it should be `verificationGasLimit` and `callGasLimit`.

Recommendation: Consider changing the definition of `accountGasLimits` to be the concatenation of `verificationGasLimit` and `callGasLimit`.

Ethereum Foundation: Fixed in PR 957.

Spearbit: Fixed.

5.4.43 ERC-4337 definition of `gasFees` uses incorrect field name

Severity: Informational

Context: erc-4337.md#L94

Description: ERC-4337's Entrypoint definition defines `gasFees` field of the Packed UserOperation struct as the concatenation of `maxPriorityFee` and `maxFeePerGas`, which is incorrect because it should be `maxPriorityFeePerGas`.

Recommendation: Consider changing the definition of `gasFees` to be the concatenation of `maxPriorityFeePerGas` and `maxFeePerGas`.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.44 ERC-4337 using `authorizer` interchangeably with `aggregator` reduces readability

Severity: Informational

Context: erc-4337.md#L126-L127

Description: ERC-4337 uses the term `authorizer` in the section on Account Contract Interface. But it uses the term `AggregatorStakeInfo` in the section on Simulation Specification. Using `authorizer` interchangeably with `aggregator` reduced readability.

Recommendation: Consider picking one term between `authorizer` and `aggregator`. Also, see related finding: `Layout, typos and outdated texts`.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.45 Mismatch exists between the ERC-4337 specification and `EntryPoint.sol` implementation of `UNUSED_GAS_PENALTY_PERCENT`

Severity: Informational

Context: EntryPoint.sol#L54-L57, EntryPoint.sol#L859-L868, erc-4337.md#L224-L225

Description: ERC-4337 specifies that:

> A penalty of `10%` (`UNUSED_GAS_PENALTY_PERCENT`) is applied on the amounts of `callGasLimit` and `paymasterPostOpGasLimit` gas that remains unused.

However, `EntryPoint.sol` refers to this `PENALTY_PERCENT` and applies it only above `PENALTY_GAS_THRESHOLD`. This illustrates a mismatch in terminology and the use of the threshold in implementation.

Recommendation: Consider:

1. Changing `PENALTY_PERCENT` to `UNUSED_GAS_PENALTY_PERCENT` in implementation for better name clarity.
2. Clarifying in the specification that this is applied only for differential above `PENALTY_GAS_THRESHOLD`.

Ethereum Foundation: Fixed in PR 882 & PR 550.

Spearbit: Fixed as recommended.

5.4.46 ERC-4337 flow chart using incorrect account names reduces readability

Severity: Informational

Context: erc-4337.md#L228

Description: The flow chart in the section "Required EntryPoint contract functionality" specifies "account" during Validations but calls it "account1" during Executions.

Recommendation: Consider using the same account name for clarity.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.47 ERC-4337 Simulate Specification section is not updated to reflect current simulation flow

Severity: Informational

Context: erc-4337.md#L317-L372

Description: ERC-4337 Simulate Specification section describes that simulation of `UserOperation` validation is performed by the bundler by making calls to `simulateValidation(userop)`. However, this now changed by PR 243: AA-440 validate using handleOps. The newly proposed approach is to make a call to the `handleOps()` method with the `UserOperation`.

Recommendation: Consider rewriting the Simulate Specification section to reflect this updated approach.

Ethereum Foundation: Fixed in PR 929.

Spearbit: Fixed as recommended.

5.4.48 ERC-4337 Reputation Rationale section incorrectly refers to "contract storage" as "memory usage"

Severity: Informational

Context: erc-4337.md#L464

Description: ERC-4337 Reputation Rationale section says: "*When staked, an entity is less restricted in its use of memory usage.*" This is incorrect because the restriction is applied to the Account Contract's storage.

Recommendation: Consider changing this to: "*When staked, an entity is less restricted in its use of contract storage.*".

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.49 ERC-4337 is missing additional security considerations

Severity: Informational

Context: erc-4337.md#L512

Description: There are more security considerations that can be added.

Recommendation: Considering adding the following:

1. Entrypoint contract: The entrypoint contract should not be upgradable, because that would potentially result in a large number of security issues.
2. Factory:.
   - The factories should check that calls to `createAccount` originate from `entryPoint.senderCreator()`.
   - Factories should be audited.
3. Paymaster:
   - Paymaster should check calls to `validatePaymasterUserOp()` and `postOp()` originate from `entryPoint`.
   - If paymasters allow anyone doing `addStake()` to the entrypoint its important that `unstakeDelaySec` isn't set, because a very high value could be set and unstake won't be possible.
   - A verifying paymaster should prevent replay attacks by hashing `block.chainid` and `address(this)` and nonces into data that is verified by a signature.
   - Paymasters should be audited.
4. 7702 delegate contract:
   - A 7702 delegate contract should check calls to its initialize function originates from `entryPoint.senderCreator()`.
5. Aggregator:
   - Aggregators should check calls to `validateSignatures` originate from `entryPoint.senderCreator()`.
   - Aggregators should be audited.
6. Smart Contract Wallet:
   - Smart contract wallets shouldn't use/rely on `tx.origin`, because this will be the `bundler`.

- If `transient storage` is used, its important to clean it at the end of the transaction, because multiple transactions can be combined in to one bundle. This includes contracts that are called from the smart contract wallet.
- When upgrading wallets: changing to an new or different implementation, or 7702 switch to other delegate: be aware of storage conflicts. Consider using erc-7201 Namespaced Storage Layout.
- When working on multiple chains/using bridges: the smart contrat wallet might not be present on other chains and/or might not be deployable there (depending on the factory and initcode and perhaps even on the compatibility of the chain).
- If the smart contract wallet allows execution of functions without going through the `EntryPoint`, the access controls and signature validations are very important.
- Wallets and wallet modules should be audited.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.50 ERC-4337 does not have an authoritative list of trusted components

Severity: Informational

Context: erc-4337.md#L121

Description: There is no good way to identify trusted components of ERC-4337. This includes:

- Entrypoints (there are multiple releases, and they are not deployed on every chain).
- Smart contract wallets and related factories.
- Modules for smart contract wallets.
- 7702 accounts.
- Paymasters.
- Bundlers (note: there is `https://www.erc4337.io/bundlers`).

Recommendation: Initiate approaches to find trusted components, starting with entrypoints. Also consider adding a reputation system for the other components. However, this should not be a centralization factor.

Possible approaches:

- Add the entrypoint addresses to ERC-4337 text (this can't be checked onchain).
- Ethereum Name Service (ENS).
- Ethereum Attestation Service (EAS).

Other initiatives for inspiration:

- `https://uniswaphooks.com/` .
- `https://chainlist.org` .
- `https://l2beat.com` .

Ethereum Foundation: After we complete the audit, we will add the deployed EntryPoint address to this repo. The scope of the audit is the core network. The `EntryPoint` as defined (by its `CREATE2` address) is the "core protocol". Because it is an ERC, it is not hard-coded into the network as "pectra" features are.

Accounts, Paymaster should explicitly support the new entrypoint, by updating their "require(msg.sender,...)" line.

As for other entities:

- The bundlers' mempool is decentralized. End-users don't know or trust a specific `bundlers`. The most a bundler can do is not serve a `UserOp`, and then another `bundler` in the mempool will.

- Any other trust on components (accounts, paymaster) by end-users is out of the scope of this audit.

Spearbit: Acknowledged.

5.4.51 ERC-4337 is missing links to other referenced ERCs

Severity: Informational

Context: erc-4337.md#L11

Description: ERC-4337 references several other ERCs but not all references have a deep link to the ERC.

Recommendation: Check and update the references.

Ethereum Foundation: Solved in several PRs.

Spearbit: Verified.

5.4.52 ERC-4337 sybil approach could be made clearer

Severity: Informational

Context: erc-4337.md#L459-L460

Description: Staking is used to prevent sybil attacks. However, it's not clear why slashing isn't used as with typical staking scenarios.

Recommendation: Consider explaining the approach to staking and sybil resistance in greater detail.

Ethereum Foundation: dded a short paragraph on the absence of slashing in PR 927.

Spearbit: Verified.

5.4.53 ERC-4337 invariant description could be made stronger

Severity: Informational

Context: erc-4337.md#L443

Description: ERC-4337 documents the main invariant as:

> It enforces the simple rule that only after validation is successful (and the UserOp can pay), the execution is done, and also guarantees the fee payment.

However this could be made stronger.

Recommendation: Consider changing the text to:

```
- the execution is done, ...
- the execution is done and only done once, ...
```

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended.

5.4.54 ERC-4337 use of `UserOperation` is inconsistent

Severity: Informational

Context: erc-4337.md#L216

Description: ERC-4337 most of the times uses the term `UserOperation`, but sometimes `operation` is used.

Recommendation: For consistency and easy readability, consider using `UserOperation` everywhere applicable.

Ethereum Foundation: Fixed in PR 927 and others.

Spearbit: Fixed as recommended.

5.4.55 ERC-4337 access lists are unclear

Severity: Informational

Context: erc-4337.md#L419

Description: ERC-4337 refers to `access lists`. However it is not clear what is meant in this context.

Recommendation: Consider explaining the `access lists`.

Ethereum Foundation: Fixed in PR 936.

Spearbit: Fixed by adding reference to EIP-2930 which elaborates on the concept.

5.4.56 ERC-7562 rules `EREP-020` and `EREP-030` are unclear on their terminology

Severity: Informational

Context: erc-7562.md#L258-L260

Description: ERC-7562 rules `EREP-020` and `EREP-030` define a staked factory and account as being "accountable" for breaking rules and thereafter that rule breaking "affects its reputation." But it is not clear what this accountability means and how their reputation is affected.

Recommendation: Consider explaining that "accountable" and "affects its reputation" means that:

1. Only its `opsSeen` is left unchanged so that its reputation is effectively reduced when `opsIncluded` is not incremented because of `FailedOp` revert or validation failure and subsequent dropping of that UserOp.
2. The `opsSeen` of other participating entities in UserOp is decremented by 1 (as done in `EREP-015`) so that their reputation is not affected.

In general, consider expanding on every rule in this spec and perhaps even providing a pseudo-code/algorithm so that bundler implementations know how to accurately model these rules.

Ethereum Foundation: Fixed in PR 940.

Spearbit: Fixed by replacing "accountable" in "If a staked..is used, its reputation is updated...".

5.4.57 ERC-7562 `inclusionRate` definition should say "ratio" instead of "relation"

Severity: Informational

Context: erc-7562.md#L112

Description: ERC-7562 defines `inclusionRate` as "Relation of `opsIncluded` to `opsSeen`." However, it may not be obvious that the relation really means ratio.

Recommendation: Consider defining `inclusionRate` as "Ratio of `opsIncluded` to `opsSeen`.

Ethereum Foundation: Fixed in PR 939.

Spearbit: Fixed as recommended.

5.4.58 ERC-7562 does not justify the values of its constants

Severity: Informational

Context: erc-7562.md#L64-L79

Description: ERC-7562 introduces several constants such as `MIN_UNSTAKE_DELAY` and `MIN_STAKE_VALUE` for staked entities, `THROTTLING_SLACK`, `BAN_SLACK` and other related ones for throttling and banning entities. The default values of these constants are provided without justification/rationale. Bundler implementers hardcoding or deviating from these values may see unexpected behavior if they do not understand the rationale behind these values.

Recommendation: Consider providing justification for these constant values so that bundler implementations can rationalize and perhaps adapt to different environments or chains.

Ethereum Foundation: Fixed in PR 958.

Spearbit: Fixed as recommended.

5.4.59 Consider renaming `preVerificationGas`

Severity: Informational

Context: erc-4337.md#L68

Description: In ERC-4337, we include the `preVerificationGas` field in the `UserOperation` to cover "Extra gas to pay the bundler". This field is used not only prior to verification, but throughout the execution of a `UserOperation`, to cover any static gas costs not covered by the other gas limit fields: `callGasLimit`, `verificationGasLimit`, `paymasterVerificationGasLimit`, and `paymasterPostOpGasLimit`. For example, we use `preVerificationGas` to cover refunding additional excess `prefund` amounts.

Since this field is used to cover static gas costs not just in the pre-verification stage, it should be renamed to better reflect its actual usage, improving readability and maintainability.

Recommendation: Consider renaming `preVerificationGas` to something more relevant, e.g. `staticGasLimit`.

Ethereum Foundation: There is a number of people and tutorials already familiar with our peculiar parameter naming, so I believe at this point any unnecessary renames will bring more confusion than clarity.

Spearbit: Acknowledged.

5.4.60 Undefined constants

Severity: Informational

Context: erc-4337.md#L123

Description: In ERC-4337, we reference the `SIG_VALIDATION_FAILED` constant:

> MUST validate that the signature is a valid signature of the `userOpHash`, and SHOULD return SIG_-VALIDATION_FAILED (and not revert) on signature mismatch. Any other error MUST revert.

However, we never define this constant in the spec. Looking at the implementation, in `Helpers.sol`, we can see that both `SIG_VALIDATION_FAILED` and `SIG_VALIDATION_SUCCESS` are defined as follows:

```
uint256 constant SIG_VALIDATION_FAILED = 1;
// ...
uint256 constant SIG_VALIDATION_SUCCESS = 0;
```

Recommendation: Define `SIG_VALIDATION_FAILED` and `SIG_VALIDATION_SUCCESS` in ERC-4337.

Ethereum Foundation: Fixed in PR 936.

Spearbit: Fixed as recommended.

5.4.61 ERC-7562 reputation definitions and calculations are unclear

Severity: Informational

Context: erc-7562.md#L115-L127

Description: ERC-7562 section on "Reputation Definitions" and "Reputation Calculation" is minimal and therefore may be unclear to the bundle implementations on the rationale for calculations. Any errors in these calculations may lead to network griefing DoS attacks by malicious entities abusing incorrect implementations of these reputation algorithms.

For example, it is not clear how: "*To help make sense of these params, note that a malicious paymaster can at most cause the network (only the p2p network, not the blockchain) to process `BAN_SLACK * MIN_INCLUSION_RATE_DENOMINATOR / 24` non-paying ops per hour.*" this works.

Recommendation: Consider expanding these Reputation sections with clearer descriptions, reference algorithms and example scenarios to facilitate correct implementations.

Ethereum Foundation: Fixed in PR 958.

Spearbit: Fixed as recommended.

5.4.62 ERC-7562 `EREP-010` clarity can be improved

Severity: Informational

Context: erc-7562.md#L253-L254

Description: It is not clear what ERC-7562 `EREP-010` means by: "*the mempool must maintain the total gas `UserOperations` using this `paymaster` may consume.*".

Recommendation: Consider rephrasing this to: "*Bundler should not accept a new `UserOperation` with a `paymaster` to the mempool, unless the paymaster's deposit can cover the total cost of all its accepted `UserOperations` at the current gas price.*".

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed with "*`[EREP-010]` For each `paymaster`, the bundler must track the total gas `UserOperations` using this `paymaster` may consume.*".

5.4.63 ERC-7562 references to "throttled" should also include "banned" for clarity

Severity: Informational

Context: erc-7562.md#L137, erc-7562.md#L275, erc-7562.md#L301

Description: Some ERC-7562 references to entities being throttled should include "banned" to clarify that throttled entities may be further banned if they continue decreasing their reputation as per any of the described rules.

Recommendation: Consider saying "throttled/banned" instead of "throttled" where appropriate.

Ethereum Foundation: Fixed in PR 928.

Spearbit: Fixed as recommended.

5.4.64 ERC-4337 is missing the definition of beneficiary address used in `handleOps()` and `handleAggregatedOps()`

Severity: Informational

Context: erc-4337.md#L102, erc-4337.md#L226

Description: The core interfaces of the `EntryPoint` contract are `handleOps()` and `handleAggregatedOps()`, which both include a `beneficiary` address parameter. While there are references to `beneficiary` in some parts, a clear definition is missing.

Recommendation: Consider adding a definition of beneficiary to clarify that it is the address that will be paid with all the gas fees collected during the execution of the bundle.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended by adding: "*The beneficiary is the address that will be paid with all the gas fees collected during the execution of the bundle.*".

5.4.65 `verificationGasLimit` is re-used multiple times for gas limit calculations

Severity: Informational

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

Description: Bundlers call `handleOps()/handleAggregatedOps()` to execute user operations, covering gas fees upfront in exchange for reimbursement (plus a premium) from either the user's smart account (`sender`) or a specified `paymaster`. The required amount is calculated as follows:.

```
function _getRequiredPrefund(
    MemoryUserOp memory mUserOp
) internal virtual pure returns (uint256 requiredPrefund) {
    unchecked {
        uint256 requiredGas = mUserOp.verificationGasLimit +
            mUserOp.callGasLimit +
            mUserOp.paymasterVerificationGasLimit +
            mUserOp.paymasterPostOpGasLimit +
            mUserOp.preVerificationGas;

        requiredPrefund = requiredGas * mUserOp.maxFeePerGas;
    }
}
```

A key variable in this calculation is `mUserOp.verificationGasLimit`. While it's included once in the `requiredGas` computation, it is referenced three separate times for gas limit enforcement:

1. `_validatePrepayment`:

   ```
   unchecked {
       if (preGas - gasleft() > _getVerificationGasLimit(verificationGasLimit)) {
           revert FailedOp(opIndex, "AA26 over verificationGasLimit");
       }
   }
   ```

2. `_createSenderIfNeeded`:

   ```
   address sender1 = senderCreator().createSender{
       gas: opInfo.mUserOp.verificationGasLimit
   }(initCode);
   ```

3. `_callValidateUserOp`:

   ```
   uint256 gasLimit = opInfo.mUserOp.verificationGasLimit;
   address sender = opInfo.mUserOp.sender;
   bool success;
   assembly ("memory-safe") {
       success := call(gasLimit, sender, 0, add(callData, 0x20), mload(callData), 0, 32)
   }
   ```

The issue arises because `_validatePrepayment` accounts for the gas used in `_createSenderIfNeeded` and `_callValidateUserOp`. This means the user must specify a higher `verificationGasLimit` than actually needed, defeating the purpose of limiting gas usage for these external calls. Consequently, excess gas may be consumed by the factory or `validateUserOp`.

Recommendation: To improve gas efficiency and prevent overestimation, consider introducing dedicated variables in `PackedUserOperation` to separately define gas limits for `_createSenderIfNeeded` and `_callValidateUserOp`. This would allow more precise control over each operation' s gas consumption.

Ethereum Foundation: Acknowledged.

> The user is expected to provide a sufficient `verificationGasLimit` and `paymasterVerificationGasLimit`, so that the validation doesn't end in revert. I think that the AA36 require statement covers the

> gas check sufficiently, so that we don't have to provide the gas limit to the `validatePaymasterUserOp` directly. However, then it means it depends on the total transaction gas limit, and adds another non-deterministic factor to the validation, so I don't see the benefit of removing it or adding another gas parameter here.

Spearbit: Acknowledged.

5.4.66 `paymasterVerificationGasLimit` is re-used twice for gas limit calculations

Severity: Informational

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

Description: Similarly to "`verificationGasLimit` is re-used multiple times for gas limit calculations", the following issue exists in `_validatePaymasterPrepayment`:

```
uint256 pmVerificationGasLimit = mUserOp.paymasterVerificationGasLimit;
try
    IPaymaster(paymaster).validatePaymasterUserOp{gas: pmVerificationGasLimit}(
        op,
        opInfo.userOpHash,
        requiredPreFund
    )
returns (bytes memory _context, uint256 _validationData) {
    context = _context;
    validationData = _validationData;
} catch {
    revert FailedOpWithRevert(opIndex, "AA33 reverted", Exec.getReturnData(REVERT_REASON_MAX_LEN));
}
if (preGas - gasleft() > _getVerificationGasLimit(pmVerificationGasLimit)) {
    revert FailedOp(opIndex, "AA36 over paymasterVerificationGasLimit");
}
```

Here, `pmVerificationGasLimit` is used twice: once for the call to `validatePaymasterUserOp` and again to measure the total gas used by `_validatePaymasterPrepayment`. If users set a strict gas limit that is insufficient for both the paymaster call and the additional logic in `_validatePaymasterPrepayment`, the validation will revert with "AA36 over paymasterVerificationGasLimit".

Recommendation: To prevent unnecessary reverts, consider introducing a separate variable in `PackedUserOperation` to specifically limit the gas allocated for the `validatePaymasterUserOp` call.

Ethereum Foundation: Acknowledged.

> The user is expected to provide a sufficient `verificationGasLimit` and `paymasterVerificationGasLimit`, so that the validation doesn't end in revert. I think that the AA36 require statement covers the gas check sufficiently, so that we don't have to provide the gas limit to the `validatePaymasterUserOp` directly. However, then it means it depends on the total transaction gas limit, and adds another non-deterministic factor to the validation, so I don't see the benefit of removing it or adding another gas parameter here.

Spearbit: Acknowledged.

5.4.67 ERC-4337 `userOpHash` description does not consider EIP-7702

Severity: Informational

Context: Eip7702Support.sol#L19-L29, erc-4337.md#L117

Description: ERC-4337 describes `userOpHash` being a hash over the userOp (except signature), entryPoint and chainId. However, it does not explain the EIP-7702 flow where the alternate value used depends on the return value of `_getEip7702InitCodeHashOverride`, which accounts for the 7702 delegate address.

Recommendation: Consider describing the EIP-7702 alternate flow for better clarity.

Ethereum Foundation: Fixed in PR 927.

Spearbit: Fixed as recommended by adding: "*Additionally, the `UserOperation` hash calculation is updated to include the desired EIP-7702 delegation address.*" to the "Support for EIP-7702 authorizations" section.

5.4.68 ERC-7562 explanation of spam/spamming/spammer can be improved to prevent different interpretations leading to network splits

Severity: Informational

Context: erc-7562.md#L54-L55, erc-7562.md#L62, erc-7562.md#L151-L154, erc-7562.md#L332-L335, erc-7562.md#L362-L367, erc-7562.md#L396-L397

Description: ERC-7562 has several references to spam/spamming/spammer. However, it does not sufficiently explain this terminology, what constitutes spam, how does a bundler mark/become a spammer, how spamming is related to reputation etc. This may lead to bundler implementations that have different interpretations for these terms and lead to network splits because of these differences.

Recommendation: Consider collating a section specifically on spam/spamming/spammer to explain their details and provide algorithms/pseudo-code to illustrate the logic.

Ethereum Foundation: Fixed in PR 958.

Spearbit: Fixed by defining a Spammer in "Validation Rules" Section.

5.4.69 Support for pre EIP-1559 networks is unnecessary

Severity: Informational

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

Description: `getUserOpGasPrice()` has the below logic to support pre EIP-1559 networks:

```
if (maxFeePerGas == maxPriorityFeePerGas) {
    //legacy mode (for networks that don't support basefee opcode)
    return maxFeePerGas;
}
```

This is unnecessary because the current implementation assumes support for EIP-1153 and EIP-7702, which succeed EIP-1559 and so such networks can be assumed to also support EIP-1559.

Recommendation: Consider removing the above logic which is not required now.

Ethereum Foundation: Fixed in PR 561.

Spearbit: Fixed as recommended.

5.4.70 `Helpers.sol` can have MIT license instead of GPL

Severity: Informational

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

Description: While `EntryPoint` and other core contracts have `// SPDX-License-Identifier: GPL-3.0`, other libraries and utilities have `// SPDX-License-Identifier: MIT` for unconstrained usage. However, `Helpers.sol` unnecessarily has the GPL license.

Recommendation: Consider changing `Helpers.sol` to `// SPDX-License-Identifier: MIT`.

Ethereum Foundation: Fixed in PR 558.

Spearbit: Fixed as recommended.

5.4.71 Front runners can cause flawed factory contracts a reputation damage and revert bundles

Severity: Informational

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

Description: User operations (`UserOps`) can deploy and initialize accounts when the operation includes an `initCode`. However, `_createSenderIfNeeded` will revert the entire bundle if the specified account already contains code:.

```
function _createSenderIfNeeded(
    uint256 opIndex,
    UserOpInfo memory opInfo,
    bytes calldata initCode
) internal virtual {
    if (initCode.length != 0) {
        address sender = opInfo.mUserOp.sender;
        if (Eip7702Support._isEip7702InitCode(initCode)) {
            if (initCode.length > 20) {
                // Already validated as an EIP-7702 delegate (and has code)
                senderCreator().initEip7702Sender(sender, initCode[20:]);
            }
            return;
        }
        if (sender.code.length != 0) // audit-note
            revert FailedOp(opIndex, "AA10 sender already constructed");
```

If the factory contract's creation function is publicly callable, frontrunners can exploit this by detecting a pending user operation and preemptively deploying the account with identical parameters via a standard ETH transaction. This results in two key issues:.

1. Bundle Reversion & Gas Griefing -- The transaction will revert, potentially wasting gas for the bundler.
2. Factory Reputation Damage -- The factory contract will be penalized for the revert, impacting its reputation.

Recommendation: To mitigate this risk, we strongly recommend factory developers ensure that account creation functions are **not publicly callable**. This potential vulnerability should be clearly documented for developers of factory contracts.

Ethereum Foundation: Solved in PR 927.

If you have a flawed factory, it should suffer reputation damage. That's the intended behavior we believe. We did add a security consideration for that in 4337:

> All `factory` contracts MUST check that all calls to the `createAccount()` function originate from the `entryPoint.senderCreator()` address.

Spearbit: Verified.

5.4.72 ERC-4337 has stale references to EIP-3074

Severity: Informational

Context: erc-4337.md#L24-L33

Description: ERC-4337 has three references to EIP-3074 which has now been superceded by EIP-7702 for Pectra upgrade and are therefore stale in this context.

Recommendation: Consider replacing the references to EIP-3074 with EIP-7702 by adding appropriate contexts.

Ethereum Foundation: Fixed in PR 911.

Spearbit: Fixed as recommended.

5.4.73 Suggested Improvements to ERC-4337 documentation

Severity: Informational

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

Description: While reading ERC-4337, we identified several areas that could be clarified or improved:

1. The diagram of user operation simulations should be updated to accurately reflect the actual function calls in the simulation process. Specifically, it should illustrate the initial basic sanity checks, followed by `simulateValidation()`, and only then `handleOps`.
2. Consider conducting an actor-based threat modeling for the different actors involved, namely: bundler, paymaster, factory, aggregator, and staked account. Structuring the ERC around potential threats and attack vectors for each actor, along with the corresponding mitigations, would improve clarity and security considerations.
3. erc-4337.md#L312: Consider explicitly stating that the restrictions on user operations apply only to the validation phase of the operation and do not affect the execution phase.
4. Provide a clear explanation of why `handleOps()` and `handleAggregatedOps()` are structured into distinct validation and execution phases.
5. erc-4337.md#L391: Consider using the term "bundle" instead of "batch" for consistency and clarity.

Ethereum Foundation: Solved in the following way:

1. We removed the call to `simulateValidation()`, instead we now call `handleOps(userOp)` in view mode to simulate it.
2. Not updated in the ERC.
3. We edited the simulation rationale in ERC4337, and added this explanation as well in ERC7562. ERC4337 links to ERC7562 in the simulation section for more details.
4. Partially addressed in 7562 since it elaborates on the validation rules.
5. Solved in PR 957.

Spearbit: Verified.

5.4.74 Enforce stricter fee per gas values

Severity: Informational

Context: erc-4337.md#L301

Description: We note in ERC-4337 that the `maxFeePerGas` and `maxPriorityFeePerGas` values provided along with a `UserOperation` must be "above a configurable minimum value that the client is willing to accept. At the minimum, they are sufficiently high to be included with the current `block.basefee`.".

It's important that these values are above the `block.basefee` as if they're below it, the bundler must take a loss to include these operations in a bundle. Realistically, we would expect the bundler to exclude them, but it may cost them some unnecessary computation in the process.

While the wording of the above quote isn't entirely clear, it seems to suggest that the `maxFeePerGas` and `maxPriorityFeePerGas` should be at least as high as the *most recent* `block.basefee`. Since the `block.basefee` can increase from block to block, we may have a sufficiently high `maxFeePerGas` and `maxPriorityFeePerGas` for the *most recent* block, while not being sufficient for the *upcoming* block, potentially causing the bundler to perform validation logic on a `UserOperation` which will not be accepted.

Recommendation: Consider changing the text to:

```diff
- At the minimum, they are sufficiently high to be included with the _current_ block.basefee.
+ At the minimum, they are sufficiently high to be included with the _upcoming_ block.basefee.
```

Ethereum Foundation: Fixed in PR 956.

Spearbit: Fixed as recommended.