OpenZeppelin | security

EVM Emulator and Semi-abstracted Nonces Update Audit

April 15, 2025

Table of Contents

Summary

Type	L2 Protocol
Timeline	From 2025-03-20 To 2025-03-28
Languages	Solidity + Yul

Total Issues	17 (15 resolved, 1 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	4 (4 resolved)
Notes & Additional Information	11 (9 resolved, 1 partially resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the pull request #1359 of the matter-labs/era-contracts repository at commit cc1619c.

In scope were the following files:

```
system-contracts
├── contracts
│   ├── Constants.sol
│   ├── ContractDeployer.sol
│   ├── EvmEmulator.yul
│   ├── EvmGasManager.yul
│   ├── NonceHolder.sol
│   ├── SystemContractErrors.sol
│   └── interfaces
│       ├── IContractDeployer.sol
│       └── INonceHolder.sol
└── evm-emulator
    ├── EvmEmulator.template.yul
    ├── EvmEmulatorFunctions.template.yul
    ├── EvmEmulatorLoop.template.yul
    └── calldata-opcodes
        └── RuntimeScope.template.yul
```

Note: Only the changes introduced in the pull request were audited. The full content of the listed files was not reviewed in its entirety.

System Overview

The audited pull request can be split into two different projects:

- Implementation of semi-abstracted nonces on the system contracts
- EVM Emulator updates.

EVM Emulator updates

Enhancing Efficiency with Pointer-based Bytecode Handling

After the changes introduced in this pull request, the `EvmEmulator` no longer copies EVM bytecodes during calls; instead, it reads them directly by pointers. This modification leverages pointers more actively, resulting in optimization improvements. By eliminating the need to copy bytecodes and utilizing pointers, the emulator enhances its efficiency, reduces memory usage, and streamlines bytecode handling.

Support for `modexp` Precompile

This pull request adds support for the `modexp` precompile in the EVM emulator and includes the implementation of gas calculations based on the input values, following the specifications of EIP-2565.

Semi-Abstracted Nonces Implementation

Requiring a single sequential nonce value is limiting the sender's ability to define their custom logic in regards to transaction ordering. In particular, ZKsync SSO's session module requires the ability to send multiple transactions in parallel without any of them overriding or cancelling the other ones.

One key change introduced in this pull request is the support for EIP-4337's semi-abstracted nonces. Before this change, accounts across the ZKChains could opt for two modes of nonce ordering: sequential and arbitrary. After this pull request, the arbitrary ordering has been deprecated and the sequencial one has been upgraded into `KeyedSequential`.

This new ordering type, allows for parallel transactions to be executed without clashing among each other, by splitting the full `uint256` nonce field into two values: a 192-bit `key` followed by a 64-bit `sequence`. Given the same `key`, the `sequence` field follows the classical sequential order, and `userOperation`s must be executed in strictly sequential order. However, multiple `key`s can be used in parallel without affecting each other.

In order so support this change, the old feature to set values under nonces (via the `setValueUnderNonce` function) has been removed. This feature allowed specific nonce invalidation by setting a value within them in a mapping.

One thing to note is that the keyed sequential ordering is backwards compatible, so all the sequential ordering accounts are treated as having been using the `key` with value zero up until now. Updating the nonce ordering is not possible anymore, and `KeyedSequential` is the default value on account creation.

Integration Considerations

This change introduces several considerations that integrators should keep in mind, such as:

- The nonce can be increased by an arbitrary value up to 2^64 through the `increaseMinNonce` function, which means that someone could theoretically reach the max nonce by performing 2^32 calls to increase their nonce by exactly 2^32.
- Before this change, the maximum theoretical nonce was 2^128, while now it is 2^64 per key.
- If an account is configured to use `Arbitrary` ordering before the update is deployed, there will not be possible for this account to migrate to `KeyedSequential`. At the time of the audit, the Matter Labs team confirmed there were zero accounts using such ordering.
- There are currently 3 different mappings tracking the nonce system. One of them is deprecated, since it kept track of nonce invalidations by setting values under them, but it is still present in the codebase. The second one keeps track of nonces with key set to zero. The last one keeps track of the different sequences of nonces per non-zero key.
- No module in the protocol is currently using the keyed nonce specific functions.
- Every new account after this update, will have by default `KeyedSequential` ordering with no way to update to any other.

Additionally, the `SsoAccount` contract currently uses the `incrementMinNonceIfEquals` method to increase the nonce after each `Transaction`. However, this method only allows the key to be zero, which means that the `SsoAccount` contract will not be able to leverage

the new keyed-nonces mechanism that would allow sending multiple `Transaction`s without them reverting due to the same nonce.

Consider updating the `SsoAccount` contract to use the new `incrementMinNonceIfEqualsKeyed` method.

Security Model and Trust Assumptions

During the audit, the following trust assumption was made based on the changes in this PR:

- **LLVM Compiler Intrinsics**: The function calls within the `verbatim` statements, such as `active_ptr_swap`, `active_ptr_data_load`, and `return_data_ptr_to_active`, belong to the LLVM compiler context. It is assumed that these intrinsics are correctly implemented and secure.

Critical Severity

C-01 Byte-to-Bit Mismatch in Shift Operations

In both `modexpGasCost` and `mloadPotentiallyPaddedValue` functions, the code calculates shift amounts in bytes but uses them directly with EVM shift instructions, which operate on bits. This results in incomplete or inaccurate shifts, as 1 byte equals 8 bits. Without converting byte-based values into their bit equivalents, the shift operations behave incorrectly.

This mismatch has several negative effects:

- **Distorted Parameter Reads**: When used to trim or isolate components like the base, exponent, or modulus, incorrect shifts may leave behind unintended bits. This can lead to inflated sizes, skewed gas calculations, or corrupted numerical values.
- **Exploitability Risk**: Malicious users may supply inputs that trigger these incorrect shifts, potentially manipulating gas costs or bypassing boundary checks, leading to undefined or exploitable behavior.
- **Incorrect Memory Interpretation**: Code paths intended to mask or sanitize specific bytes may instead leave residual bits intact. This can cause logical errors when interpreting memory content.
- **Numerical Instability**: Misaligned shift results can cause values to overflow or underflow in downstream logic. For instance, malformed bit-lengths derived from exponent parsing may cause loops to run excessively or insufficiently.

Consider ensuring that every shift amount derived from a byte difference is multiplied by 8 before applying any shift operation. This guarantees alignment with EVM's bit-level shift semantics and avoids the wide range of downstream issues stemming from partial shifts.

Update: *Resolved in pull request #1383.*

High Severity

H-01 Inner Variable Shadowing Causes Incorrect Return in `mloadPotentiallyPaddedValue`

The helper function `mloadPotentiallyPaddedValue` is intended to read a 32-byte word from memory and zero out any bytes that lie beyond a specified memory boundary. However, due to improper use of a `let` declaration inside an `if` block, the adjusted value is not actually returned.

```
function mloadPotentiallyPaddedValue(index, memoryBound) -> value {
    value := mload(index)

    if lt(memoryBound, add(index, 32)) {
        memoryBound := getMax(index, memoryBound)
        let shift := sub(add(index, 32), memoryBound)
        let value := shl(shift, shr(shift, value)) // inner `value` shadows outer
    }
}
```

In the `if` block, a new local variable named `value` is declared using `let`, which shadows the outer `value` that is the function's return variable. As a result, any transformation applied within the block affects only the inner `value` and not the function's output. This leads to the function returning the original unmodified result of `mload(index)`, even when part of the read spans beyond the specified memory region.

As an additional observation, while variable shadowing is disallowed in Yul, the current compiler does not enforce this rule and fails to emit an error. This leads to subtle logic bugs such as this one, where the code appears correct but behaves unexpectedly due to silent shadowing.

This issue has downstream implications for gas cost estimation in the `modexpGasCost` function, which relies on `mloadPotentiallyPaddedValue` to extract bounded parameters. If those values are not correctly adjusted, the computation proceeds with inaccurate inputs:

- **Incorrect parameter sizes**: When memory bounds are exceeded, out-of-bound bytes remain in the value, leading to misinterpreted sizes.
- **Wrong exponent iteration count**: An incorrect `Esize` affects the bit length estimation, skewing the iteration logic.

- **Incorrect gas metering**: The gas cost may be significantly under- or over-estimated, defeating the purpose of precise metering and potentially leading to exploitability or denial of service.

Consider assigning the adjusted value directly to the return variable, avoiding the use of a shadowing `let` declaration.

Update: *Resolved in pull request #1384.*

Low Severity

L-01 Missing Docstrings

Docstrings are essential to improve code readability and maintenance. Providing clear descriptions of contracts, functions (including their arguments and return values), events, and state variables helps developers and auditors better understand code functionality and purpose.

Multiple instances of missing or incomplete docstrings were identified across several contracts, such as:

- The `ContractDeployer` contract, where not all functions include docstrings for their arguments and return values.
- The `INonceHolder` interface, which only describes function names without documenting arguments and return values.
- The `IContractDeployer` interface, which lacks documentation for several functions, events, arguments, and return values.

Consider thoroughly documenting all contracts, functions, events, and relevant state variables using clear, descriptive docstrings. Documentation should adhere to the Ethereum Natural Specification Format (NatSpec) standard to enhance readability, support auditing efforts, and improve long-term maintainability.

Update: *Resolved in pull request #1399 at commits 250af39 and ae0a314.*

L-02 Deprecation of Arbitrary Ordering Is Not Explicit

The new implementation of the `ContractDeployer` contract prevents accounts from updating their nonce ordering system. Additionally, `KeyedSequential` is specified as the default ordering, which makes the `Arbitrary` ordering option fully deprecated.

However, there are still places that do not reference this deprecation of the `Arbitrary` ordering which could cause confusion. In particular:

- The documentation and name of the element in the `enum` corresponding to the `Arbitrary` type in the `IContractDeployer` interface.
- The broad documentation of the contract when referencing the nonce ordering.

Consider updating the documentation and `enum` value to properly reflect that this nonce ordering system is now deprecated in order to improve code readability, avoid confusion, and make the current design choice explicit. Additionally, even if there is not any account with `Arbitrary` ordering configured, there is a chance that one could set it before this code is deployed. Consider adding a function that would allow any account configured to use `Arbitrary` ordering to strictly migrate to `KeyedSequential`. This would provide these accounts with a way to correctly migrate in case they unknowingly update to `Arbitrary` before the update.

Update: *Resolved in pull request #1387 at commit 051b360. The Matter Labs team stated:*

> *Migrating from `Arbitrary` ordering back to `KeyedSequential` is forbidden due to the assumptions that `KeyedSequential` ordering makes. Namely, if nonce value for nonce key K is V, the assumption is that none of the values above V are used. This assumption would break if account migrates from `Arbitrary` ordering.*

L-03 Lack of Input Validation

The `_value` argument from the `increaseMinNonce` function in the `NonceHolder` contract lacks input validation, and it should be strictly greater than zero when called.

Consider implementing a validation check to ensure `_value > 0` in order to prevent unexpected behavior.

Update: *Resolved in pull request #1388 at commits aa15081 and a44fc21. The* `NonceIncreaseError` *custom error has also been modified to inform about the minimum possible value too, which is 1.*

L-04 Unreachable Code

The helper function `MAX_MODEXP_INPUT_FIELD_SIZE` restricts each input field (`Bsize`, `Esize`, `Msize`) to a maximum of 32 bytes. If any of these exceed 32, `modexpGasCost` exits early by returning `MAX_UINT64()`.

Despite this restriction, the function contains a `switch` that branches on whether `Esize > 32`, with logic intended to handle larger exponents. However, this branch is currently unreachable due to the enforced limit, making it effectively dead code.

This may be intentional for future-proofing, but as it stands, the logic adds unnecessary complexity.

Consider adding clear documentation explaining why this code path is currently unreachable, and under what future conditions it may become relevant.

Update: *Resolved in pull request #1385. The Matter Labs team stated:*

> *This is intentional indeed -* `MAX_MODEXP_INPUT_FIELD_SIZE` *can be arbitrary, and in the current version it is 32 bytes. However, more comments have been added.*

Notes & Additional Information

N-01 Inconsistent Interface Between Sibling Repositories

The current pull request #1359 in the `era-contracts` repository is linked to pull request #3646 in the `zksync-era` repository through pull request #1299 in the former. Both

repositories include changes to the `INonceHolder` interface, but the modifications are not aligned:

- The pragma directive is floating in both repositories but uses a different version as the base.
- The `ValueSetUnderNonce` event and the `isNonceUsed` function are not present in the `zksync-era` version of the interface.
- Conversely, the `setValueUnderNonce` and `getValueUnderNonce` functions exist only in the `zksync-era` repository and have been removed from the `era-contracts` version.

Although the `zksync-era` version is intended for testing purposes, maintaining consistency across both repositories is important to ensure correctness, reduce confusion, and preserve testing robustness.

Consider aligning the `INonceHolder` interface definitions in both repositories, or clearly documenting their divergence if intentional.

Update: *Resolved at commit e911061 on the zksync-era repository. Now both repositories are consistent on the `INonceHolder` interface definition.*

N-02 Misleading Documentation

Throughout the codebase, there are instances where existing comments may be misleading or outdated. In particular:

- The default ordering system has been updated from `Sequential` to `KeyedSequential`, but several comments still reference the previous default. This inconsistency appears in the `ContractDeployer.sol` contract on lines 312, 437, and 465.
- On line 26 of `ContractDeployer.sol`, the comment states that the `AccountInfo` value will be zero for EOAs and simple contracts. This could be clarified to specify that a zero value corresponds to the default `None` account abstraction version and the `KeyedSequential` nonce ordering.

Consider updating these inline comments to accurately reflect the current system behavior. Doing so will improve readability and reduce the risk of confusion during future development or review.

Update: *Resolved in pull request #1399 at commit ffcf289.*

N-03 Mismatch Between Interface and Implementation

Throughout the codebase, there are some mismatches between interfaces and their associated implementations.

The `updateNonceOrdering` function within the `IContractDeployer` interface and its implementation in the `ContractDeployer` contract present the following differences:

- The interface contains a named argument, while the implementation omits it to indicate that the function should not be used, as it will always revert.
- The implementation docstring states that the nonce ordering system cannot be updated, while the interface suggests that it can.

The `NonceHolder` contract introduces a new `getKeyedNonce` function that is not included in the associated `INonceHolder` interface. Additionally, the function ordering in the interface does not match the implementation.

Consider applying the following consistency improvements:

- Remove the parameter name from the interface version of `updateNonceOrdering` to align with the implementation.
- Update the interface docstring to reflect that the function is deprecated and will always revert.
- Add the `getKeyedNonce` function to the `INonceHolder` interface.
- Align function ordering in interfaces with the corresponding implementation contracts.

These changes would help reduce confusion and avoid unexpected usage patterns across the codebase.

Update: *Partially resolved in pull request #1392 at commit 58acf0c. The Matter Labs team stated:*

> *The bullet points 1-3 were fixed. Reordering function declarations introduces merge conflicts that are not worth the minor readability gain.*

N-04 Inconsistent Handling of Nonce Types

Throughout the codebase, there are instances where nonce values are handled inconsistently, leading to ambiguity in interpretation and usage. In particular:

- The `getKeyedNonce` function from the `NonceHolder` contract returns data from the `rawNonce` mapping when the key is zero, but returns data from the `keyedNonces` mapping when a non-zero key is provided. However, in the `isNonceUsed` function, this logic is not consistently applied—passing a zero key results in querying both the `keyedNonces` and `rawNonce` mappings, instead of only `rawNonce`.
- The `incrementMinNonceIfEquals` function uses the `_expectedNonce` input as both a combined keyed-type nonce and a `minNonce`-type nonce, resulting in dual interpretation of the same value.

Although these inconsistencies do not currently introduce security vulnerabilities, having multiple ways to interpret or validate the same data increases the potential for future bugs, reduces clarity, and complicates maintenance.

Consider unifying the logic for handling nonce types across functions to improve consistency and code readability.

Update: *Resolved in pull request #1395 at commit e04af13 and in pull request #1403, at commit 6eb1fb2.*

N-05 Function Visibility Overly Permissive

Throughout the codebase, there are various functions with visibility levels that are more permissive than necessary:

- The `getKeyedNonce` function in `NonceHolder.sol` is marked `public` but could be limited to `external`.
- The `getRawNonce` function in `NonceHolder.sol` is marked `public` but could be limited to `external`.
- The `increaseMinNonce` function in `NonceHolder.sol` is marked `public` but could be limited to `external`.
- The `_splitRawNonce` function in `NonceHolder.sol` is marked `internal` but could be limited to `private`.

Consider restricting function visibility to the minimum necessary in order to better reflect intended usage and potentially reduce gas costs.

Update: *Resolved in pull request #1391.*

N-06 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to report vulnerabilities. This practice allows the code owners to define a preferred communication channel for responsible disclosure, reducing the risk of miscommunication or missed reports. Additionally, in cases where third-party libraries are used, maintainers can easily reach out with mitigation guidance if needed.

Throughout the codebase, there are contracts that do not include a security contact:

- The `IContractDeployer` interface.
- The `INonceHolder` interface.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` tag is recommended, as it has been adopted by tools like OpenZeppelin Wizard and repositories such as ethereum-lists.

Update: *Resolved in pull request #1399 at commit bfbe2a8.*

N-07 Functions Updating State Without Event Emissions

Throughout the codebase, multiple instances of functions update contract state without emitting corresponding events. Examples include:

- The `increaseMinNonce` function in `NonceHolder.sol`
- The `incrementMinNonceIfEquals` function in `NonceHolder.sol`
- The `incrementMinNonceIfEqualsKeyed` function in `NonceHolder.sol`
- The `incrementDeploymentNonce` function in `NonceHolder.sol`

Consider emitting events for all state-changing operations to improve transparency, support off-chain indexing, and reduce the risk of silent state mutations that may be difficult to track or audit later.

Update: *Acknowledged, not resolved. The Matter Labs team stated:*

> *Since EVM does not emit events for nonce increments, it was decided to not emit them in `NonceHolder.sol` to not make developers rely on them. Custom AA accounts may still decide to emit them from their own contracts, if they should need them.*

N-08 Unused Event

In the `INonceHolder` interface, the `ValueSetUnderNonce` event is defined but is not used throughout the codebase.

Consider removing the unused event to improve code readability and reduce unnecessary interface clutter.

Update: *Resolved in pull request #1390.*

N-09 Redundant Return Statements

To improve the readability of the codebase, it is recommended to remove redundant return statements from functions that have named returns.

Throughout the codebase, there are multiple instances of redundant return statements. Some of them fall outside of the current audit scope; however, it is beneficial to highlight them as well:

- Line 41 from the `getAccountInfo` function in `ContractDeployer.sol` should assign to the `info` return variable.
- Line 217 from the `precreateEvmAccountFromEmulator` function in `ContractDeployer.sol` is redundant.
- Line 417 in function `_evmDeployOnAddress` in `ContractDeployer.sol` should assign the final value to the `constructorReturnEvmGas` return variable.
- Lines 473-480 from the `_performDeployOnAddressEVM` function should assign the internal function output to the `constructorReturnEvmGas` return variable in `ContractDeployer.sol`.
- Line 180 from the `getDeploymentNonce` function in `NonceHolder.sol` is redundant.

Consider removing the redundant return statement in functions with named returns to improve the readability of the contract.

Update: *Resolved in pull request #1394.*

N-10 Implicit Casting

The lack of explicit casting hinders code readability and makes the codebase hard to maintain and error-prone.

The `nonceValue` parameter is implicitly cast to `uint256` from `uint64` within the `_combineKeyedNonce` function.

Consider explicitly casting all integer values to their expected type to improve readability and reduce the risk of subtle bugs in future updates.

Update: *Resolved in pull request #1393.*

N-11 Inconsistent Variable Naming

Throughout the codebase, all variables follow the "camelCase" naming convention. However, when calculating the gas cost for the `Modexp` precompile, there are some instances where this convention is violated.

When retrieving the base, exponent, and modulus lengths in bytes, these variables are named `Bsize`, `Esize`, and `Msize`, respectively.

Consider enforcing consistency in the naming convention used across all variables by renaming these to `bSize`, `eSize`, and `mSize`.

Update: *Resolved in pull request #1386.*

Conclusion

This audit focused on the implementation of the `ModExp` precompile gas cost calculation, the adjustments in the Emulator to avoid unnecessary bytecode copying via pointer usage, and the introduction of semi-abstracted nonces in the system contracts in accordance to EIP-4337.

During the audit, one critical and one high-severity issue were identified. In addition, several issues related to optimization, insufficient checks, low test coverage and best practices were identified that, while not immediately threatening to system security, could impact performance, maintainability, and gas efficiency.

Despite these findings, communication with the team was notably fast and friendly, and the modular nature of the code indicates an overall organized approach. That said, there is considerable room for improvement in areas such as comprehensive documentation of recent changes and more robust testing strategies. Addressing these concerns will better position the project for future upgrades.