Securing The `CoinbaseSmartWallet` with EIP-7702: Challenges and Solutions

Introduction

EIP-7702 represents a significant advancement in Ethereum account functionality, enabling Externally Owned Accounts (EOAs) to transform into smart contract wallets through a mechanism called "delegation designation." This capability allows EOAs to delegate their execution logic to a designated delegate address, opening up a world of smart contract functionality to previously simple accounts. However, this powerful feature introduces complex security challenges that require careful consideration and robust solutions. We describe below the specific challenges to safely using the existing Coinbase smart wallet implementation in the context of EIP-7702 upgrades and the design solutions that make this possible.

The Challenge of Secure Initialization

The first major hurdle in implementing EIP-7702 stems from its initialization constraints. Traditional smart contract wallets, including the `CoinbaseSmartWallet`, typically handle secure initialization through atomic deployment and initialization via factory patterns. EIP-7702's design, however, does not permit execution of initialization calldata during the delegation authorization process (the comparable step to “deployment” in the traditional sense).

This limitation creates a critical vulnerability window. When an implementation contract is "deployed" to an EOA via EIP-7702, there's a period between the delegation and the initialization where this type of contract would exist in an unprotected state. During this window, an attacker could potentially front-run the initialization transaction and claim ownership of the smart account.

Note that existing Coinbase smart wallets are deployed behind an ERC-1967 proxy with a UUPSUpgradeable implementation (the actual `CoinbaseSmartWallet` logic). The code at the actual account address is a proxy, and the proxy uses a conventional storage location defined by ERC-1967 to house a pointer to its implementation logic. Our solution to the initialization vulnerability in a 7702 context stems from recognizing that the implementation logic only becomes active (and therefore dangerous) when the proxy actually establishes this pointer. Therefore, if we can’t enforce atomic *deployment* and initialization, we can instead enforce atomic *implementation setting* and initialization.

In the context of EIP-7702 the EOA itself is the natural authority over a newly delegated smart account. Our `EIP7702Proxy` contract implements an EOA-signature-based initialization process through a `setImplementation` function. Prior to setting its implementation pointer to

a potentially vulnerable implementation, this function first validates a signature from the EOA. It then leverages the `upgradeToAndCall` pattern from the UUPSUpgradeable proxy standard, allowing us to atomically set the implementation address and execute initialization (or any arbitrary) calldata in a single, authorized transaction. To reduce the likelihood of an implementation resulting in a vulnerable state after being set in the proxy, the `setImplementation` function is further parameterized by the address of an implementation-specific validator contract that implements an interface. This validator is called before execution flow terminates to evaluate whether the implementation is considered to be in a safe state.

The Storage Persistence Problem

Perhaps the most intricate challenge of EIP-7702 relates to storage management. The EOA can freely re-delegate or completely remove delegation at any time, and all delegates share the same storage address space. The management of storage collisions is already a familiar problem in the proxy design space, where mutable implementation logic is used to govern shared storage. However, in a proxy system there is usually at least a guarantee that the proxy logic itself is immutable and therefore state transitions are somewhat bounded by the logic defined on the proxy and/or each successive implementation. 7702 re-delegation introduces another layer of total mutability on top of this design space, creating a scenario where storage guarantees between delegations become almost non-existent.

A particularly challenging scenario is a delegate transition from delegate A to B and back to A again – the returning delegate cannot make assumptions about the state of its storage, which may be in an inconsistent state that would have been impossible to achieve via the logic of delegate A alone. A delegate might find its storage modified or entirely wiped by another delegate that previously controlled the account.

Our solution to this challenge operates on multiple levels. First, as described above, we protect the vulnerable smart account’s initialization process by ensuring that any implementation set through our proxy simultaneously achieves its safe initialization state. However, we recognized that this alone wasn't enough – a prior initialization signature could potentially be replayed if storage was wiped and then the original delegate reinstated since the safe initialization state would no longer be in place. This led us to implement a separate `NonceTracker` singleton contract that maintains per-account nonces used in signature validation. By enforcing nonces in the signatures and moving nonce tracking outside the account's storage space, we prevent other delegates from manipulating these security-critical values in the account storage. The NonceTracker ensures that each `setImplementation` signature can only be used once, regardless of what happens to the account's internal storage.

The above measures protect an implementation (when being used with an `EIP7702Proxy`) from becoming vulnerable to re-initialization with old signatures or via storage manipulation. However, it does not protect an implementation from other arbitrary storage disruption. In the

same way that an unwitting EOA can sign a drainer transaction, a malicious 7702 delegate authorized by an EOA could also be arbitrarily ruinous, and we consider protecting against these possible outcomes to be outside the scope of our design. We aim to protect against incidental problems that may arise in a multi-wallet, 7702-enabled ecosystem composed of well-intentioned if chaotic actors, but cannot protect against the possibilities that arise if an EOA is willing to authorize a truly malicious delegate.

Shared conventional storage locations

While all storage locations are potentially subject to interference between different delegates under EIP-7702, standardized storage slots like those defined by ERC-1967 are particularly vulnerable. This is because these slots, by virtue of being conventional standards, are likely to be used by multiple delegate implementations for the same purpose. The ERC-1967 implementation slot is an excellent example: when an account transitions between different delegates (A→B→A) where both delegates implement ERC-1967 proxy patterns, delegate B will naturally use the same storage slot that delegate A was using to store its implementation address. During its tenure, delegate B could modify or overwrite this slot, either intentionally or as a normal part of its own proxy operations. This creates a bootstrapping problem when attempting to return to delegate A, as the very mechanism for establishing the implementation (`upgradeToAndCall`, implemented on the implementation contract) may have been compromised.

Our `EIP7702Proxy` addresses this through its `setImplementation` function, which provides a persistent, implementation-independent upgrade mechanism secured by EOA signatures. This ensures that even if an intervening delegate has pointed the ERC-1967 implementation at an invalid implementation (or removed it entirely), the original EOA, after delegating back to an `EIP7702Proxy`, maintains the ability to reconfigure the implementation pointer to their chosen contract.

Token Reception and Default Implementation

A seemingly simple but crucial challenge involves token reception. An ERC-1967 proxy without an implementation cannot receive tokens that require specific receiver hooks, such as ERC-721 or ERC-1155 tokens, or even native ether through standard receive functions.

We addressed this by incorporating a default Receiver implementation as an immutable value in the EIP7702Proxy. This receiver automatically handles token reception when no ERC-1967 implementation is set, ensuring the account can always accept token transfers regardless of its delegation state.

Why We Didn't Adopt ERC-7779

During our development process, we considered ERC-7779, which proposes a standard for tracking storage usage across different delegates. While well-intentioned, this standard relies on voluntary compliance and cannot provide strong security guarantees. The proposed storage bases registry and optional `onRedelegation` function offer a "best effort" approach to storage management, but in a system where security is paramount, best efforts aren't enough.

Instead, we focused on building self-defensive mechanisms that maintain particular security properties regardless of delegation history or storage state. Our solution acknowledges the fundamental mutability of storage in EIP-7702 accounts and implements protective measures that don't rely on the cooperation of other delegates.

Conclusion

The EIP7702Proxy design allows us to continue to rely on the same `CoinbaseSmartWallet` account implementation used in traditional smart contract wallet deployments while solving the unique security challenges that arise in the context of EIP-7702. By carefully considering initialization security, the implications of storage impermanence, and seamless token handling, we've created a secure proxy for EIP-7702 smart contract wallets.