Sequence Audit Report

The modular crypto infrastructure stack that unifies wallets, 1-click cross-chain transactions, and real-time data.

Version 1.0

Prakash Yadav

October 8, 2025

Sequence Audit Report

Prakash Yadav

October 8, 2025

Prepared by: Prakash Yadav Lead Auditors: - Prakash Yadav

Table of Contents

Protocol Summary

Sequence Ecosystem Wallet is a non-custodial smart wallet designed for chains and ecosystems. It combines passkeys, social auth, timed recovery keys, and sandboxed permissions to deliver higher security with less friction.

The codebase represents the V3 implementation of this infrastructure, utilizing a minimal proxy pattern and a novel Merkle-proof based configuration approach for smart wallets. # Disclaimer

Prakash Yadav makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Scope

File
src/Factory.sol
src/Guest.sol

File
src/Stage1Module.sol
src/Stage2Module.sol
src/Wallet.sol
src/extensions/passkeys/Passkeys.sol
src/extensions/recovery/Recovery.sol
src/extensions/sessions/SessionErrors.sol
src/extensions/sessions/SessionManager.sol
src/extensions/sessions/SessionSig.sol
src/extensions/sessions/explicit/ExplicitSessionManager.sol
src/extensions/sessions/explicit/IExplicitSessionManager.sol
src/extensions/sessions/explicit/Permission.sol
src/extensions/sessions/explicit/PermissionValidator.sol
src/extensions/sessions/implicit/Attestation.sol
src/extensions/sessions/implicit/ISignalsImplicitMode.sol
src/extensions/sessions/implicit/ImplicitSessionManager.sol
src/modules/Calls.sol
src/modules/ERC4337v07.sol
src/modules/Hooks.sol
src/modules/Implementation.sol
src/modules/Nonce.sol
src/modules/Payload.sol
src/modules/ReentrancyGuard.sol
src/modules/Storage.sol
src/modules/auth/BaseAuth.sol
src/modules/auth/BaseSig.sol
src/modules/auth/SelfAuth.sol
src/modules/auth/Stage1Auth.sol

File
src/modules/auth/Stage2Auth.sol
src/modules/interfaces/IAccount.sol
src/modules/interfaces/IAuth.sol
src/modules/interfaces/ICheckpointer.sol
src/modules/interfaces/IDelegatedExtension.sol
src/modules/interfaces/IERC1155Receiver.sol
src/modules/interfaces/IERC1271.sol
src/modules/interfaces/IERC223Receiver.sol
src/modules/interfaces/IERC721Receiver.sol
src/modules/interfaces/IERC777Receiver.sol
src/modules/interfaces/IEntryPoint.sol
src/modules/interfaces/IPartialAuth.sol
src/modules/interfaces/ISapient.sol
src/utils/Base64.sol
src/utils/LibBytes.sol
src/utils/LibOptim.sol
src/utils/P256.sol
src/utils/WebAuthn.sol
Total Logic Contracts: 34

For a machine-readable version, see scope.txt

Files out of scope

File
script/**.**
src/Estimator.sol

File
src/Simulator.sol
test/**.**
Total Contracts: 45

For a machine-readable version, see out_of_scope.txt

Roles

Authorized Signers Only

Only the wallet's designated signers (meeting the required signing threshold) can execute transactions or make state changes. No external party can operate the wallet without a valid EIP-712 signature from the correct signer set. The contract enforces this by requiring a proper signature for every execute call – if the signature check fails, the transaction is rejected.

Image Hash & Configuration Integrity

Each Sequence wallet is defined by an image hash that encodes its owner configuration (the set of signer addresses and their threshold scheme). This image hash is tied to the wallet's deployment address. On deployment, the wallet contract checks that its own address was generated using the image hash (via CREATE2 with the factory) and stores this hash on-chain. Every future transaction recomputes the image hash from the current config and compares it to the stored value, ensuring the signer set or threshold cannot be tampered with undetected. In short, the wallet's address and its authorized signer configuration are cryptographically bound – any unauthorized change breaks the hash check and invalidates signatures.

Deterministic Wallet Address per Config

Given the above, a particular signer configuration always corresponds to a single unique wallet address. The factory uses the image hash as a salt to deploy the wallet, meaning the mapping between a wallet's config and its address is one-to-one. This prevents an attacker from, say, front-running the deployment of a user's wallet with a different contract – the address is predetermined by the intended signers. No two distinct configs will produce the same address, and the same config cannot be deployed twice

on the same network. After deployment, two wallets could upgrade their image hash to the same configuration.

Strict Nonce Sequencing

Sequence wallets implement a multi-space nonce system to prevent replay attacks. Each wallet has independent nonce "spaces" (to allow parallel sequence streams), and in each space the nonce must match exactly the next expected value. Nonces increment sequentially per space and cannot be reused. If a transaction's provided nonce is out of sequence for that space, it will be rejected as an INVALID_NONCE. This invariant guarantees proper ordering of transactions and that each signed transaction is unique to a single execution.

Domain-Separated Signatures

All signatures are domain-separated and network-specific. The wallet's EIP-712 signing scheme includes the current chain ID and the wallet's address in the hashed message. This means a signature intended for one particular Sequence wallet on one network cannot be replayed on a different wallet or chain. The contract explicitly pulls the chain ID in at hash time and prefixes the data with `0x19_01 || chainId || address(this)`, binding the signature to that wallet instance. This invariant protects against cross-chain or cross-contract replay of signed messages.

Privileged Operations Require Self-Call

Sensitive operations on the wallet (such as upgrading the implementation, adding/removing module hooks, or deploying new contracts from the wallet) are guarded by a modifier onlySelf. This means the function can only be called by the wallet itself (i.e. via an internal delegatecall from the wallet's own context) and never by an external EOA or unprivileged contract . For example, the updateImplementation function (used to upgrade the wallet's logic) is onlySelf, so it can only execute if initiated from an authorized wallet transaction, and cannot be invoked by an attacker directly. This invariant ensures no admin or external contract can unilaterally change the wallet's state – only the wallet's owners, via a proper signed transaction, can trigger such changes.

All External Actions Go Through execute

Users interact with their Sequence wallet exclusively via the execute function (or meta-transaction workflows that ultimately call execute). There is no alternative public method to trigger arbitrary calls from the wallet without signature verification. Even batched calls are executed internally by _execute

after the signature and nonce have been validated. This invariant means there's no "backdoor" to bypass authentication – every funds transfer or contract call from the wallet is explicitly authorized by the wallet's signers.

ERC-4337 Integration

Sequence wallets fully support ERC-4337 account abstraction by implementing the required validateUserOp interface. When a User Operation is submitted through an ERC-4337 entrypoint, the wallet validates that the sender is the configured entrypoint contract and processes the operation through the executeUserOp function. The validateUserOp function calls the wallet's own ERC-1271 isValidSignature function to validate signature correctness, where the signature is of the userOpHash rather than the Payload contents. This design protects against replay attacks by ensuring that signatures are bound to the specific User Operation hash, which includes critical fields like nonce, gas parameters, and operation data. The User Operation's data field contains a nested Sequence Payload, which gets forwarded to the wallet's selfExecute function. This selfExecute call follows the same execution flow as the standard _execute function described above. This invariant guarantees that ERC-4337 operations maintain the same security guarantees as direct wallet interactions – the account abstraction layer cannot bypass the wallet's core authentication mechanisms or authorization requirements.

Batched Transactions & Atomicity

The wallet supports batching multiple actions in a single execute call for efficiency. By default, the batch is atomic – if any call in the batch fails and is marked as critical, the entire batch will revert. However, the wallet allows certain calls to be flagged as non-critical (revertOnError = false), in which case a failure of that call will not stop the batch: it will emit a TxFailed event for that specific sub-transaction and continue with the next one. This invariant ensures that optional or best-effort operations can be attempted without jeopardizing the main transaction, while still transparently logging any failures. Importantly, a sub-call failing without revertOnError cannot corrupt subsequent calls – the revert is trapped and the wallet moves on, maintaining overall state consistency for the rest of the batch.

Contract Signers and ERC-1271

Sequence wallets can have other smart-contract wallets or contracts as signers (not just EOAs), and the wallet fully supports nested signatures via ERC-1271 and the new Sapient Signer interface. If a signer is a contract, the Sequence wallet will call that contract's isValidSignature method to confirm that the payload was approved by that contract's logic. A contract signer only counts as valid if its own internal approval check returns true, per ERC-1271. This means adding a contract (even another Sequence

wallet) as a signer does not bypass the signature requirement – it simply shifts it to that contract's own signature/approval mechanism. The system even supports multiple layers of nested Sequence wallets as signers, as covered in tests (e.g. wallets signing for wallets), all of which must resolve to true approvals. Invariantly, a signature from a contract signer is treated with the same rigor as a human signer: no contract signer can "auto-approve" transactions unless explicitly programmed to, and it cannot be used to circumvent the threshold or nonce rules.

Sapient Signers

Sapient signers represent an advanced interface designed to support more complex Sequence wallet signer configurations. While ERC-1271 can only validate a hash and signature by returning the magic value, a Sapient Signer receives the complete transaction payload and signature and is expected to return its configuration or image hash. This returned image hash is then used by the Sequence wallet to reconstruct the wallet's image hash. Since the signer's image hash is derived for every payload and signature combination, a sapient signer can counterfactually determine its image hash without relying on on-chain state. This capability is achieved by encoding the intended sapient signer configuration directly within the signature being validated. As a result, a wallet can support a specific configuration of the sapient signer without requiring updates to that contract's state, enabling more flexible and gas-efficient signer management. The sapient signer may validate the contents of the payload or signature in whichever way it deems fit.

Controlled Module Hooks

The wallet allows installing hook modules to handle specific function selectors (for example, to custom-handle incoming token transfers or to extend wallet functionality). These hooks are strictly controlled by the wallet's owners. Only one hook implementation can be registered per function signature at any time, and adding or removing a hook can only be done via a valid wallet transaction (which, as noted, requires signer authorization and onlySelf). If a hook is set for a function, the wallet's fallback will delegatecall into the hook's contract when that function is invoked; if no hook is set, such calls are simply ignored by the wallet's fallback (no action taken, aside from possibly receiving ETH). Hooks do not get to override the wallet's security model – they execute within the wallet context under the same onlySelf restrictions for any state changes. In essence, a hook can extend functionality but cannot, for example, surreptitiously initiate an execute on its own. The invariant here is that hooks augment the wallet but cannot violate its core access controls.

Non-Privileged Helper Modules

The Sequence system includes certain helper modules (e.g. Estimator, Simulator, and a Guest module for new wallets) which have no privileged rights in the protocol. These components are used for off-chain simulation, gas estimation or temporary guest session logic, and they cannot modify wallet state or perform sensitive actions. Invariants are not impacted by these modules – they operate with read-only or strictly limited scope. This means auditors and users can largely ignore these modules in terms of security critical paths, as they cannot bypass authorization or affect funds. All the critical invariants remain focused on the core wallet, its factory, and the authorized modules described above.

Executive Summary

Issues found

Category	No. of Issues
High	1
Medium	5
Low	1
Informational	6
Total	13

Findings

High

[H-1]:Using `delegatecall` in loop.

Description:

The code iterates through multiple calls and uses `delegatecall` inside the loop for calls with `delegateCall == true`. Because `delegatecall` executes code in the context of the calling contract (this contract's storage, msg.sender remains original), and certain semantics such as msg.value and gas accounting can be tricky, using delegatecall repeatedly inside a loop may lead to unexpected state interactions, reentrancy risks, and repeated accounting of msg.value or other invariants. The

report warns that the same msg.value could be “credited” multiple times if delegate-called logic treats msg.value or other re-entrancy-sensitive state incorrectly.

Impact:

If the delegated code expects to run in its own contract (not as delegatecall), state modifications will write into the caller contract storage. This can corrupt wallet state or allow unexpected modifications. If delegated code sends or relies on Ether (msg.value) behavior, repeated delegatecalls may mis-handle value accounting. Reentrancy vectors: delegatecalls that call back into the calling contract may exploit intermediate state.

Proof of Concepts:

Construct a payload with multiple calls where one call is delegateCall to an extension that writes to a storage slot in the caller (e.g., increments a counter) and another delegateCall expects the counter to be zero. Running both delegatecalls in the same execution will update storage twice (unexpected state), leading to logical errors or privilege escalation.

Illustrating delegatecall in the loop (from repo):

Calls contract:

```

if (call.delegateCall) {
  (success) = LibOptim.delegatecall(
    call.to,
    gasLimit == 0 ? gasleft() : gasLimit,
    abi.encodeWithSelector(
      IDelegatedExtension.handleSequenceDelegateCall.selector,
      _opHash,
      _startingGas,
      i,
      numCalls,
      _decoded.space,
      call.data
    )
  );
}
```

Alternatively, Create a malicious extension contract with a function that increments a storage slot at a deterministic location (same slot the wallet expects for e.g., counter). Craft a payload with two delegateCall entries to that extension — after the loop both increments are applied to the caller’s storage.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

// This is a delegatecall target. When called via delegatecall it will
```

```
// write to the caller's storage slot 0 (for demonstration).
contract MaliciousDelegate {
  function handleSequenceDelegateCall(bytes32 /*opHash*/, uint256 /*
      startingGas*/, uint256 /*i*/, uint256 /*numCalls*/, bytes32 /*
      space*/, bytes calldata /*data*/) external {
    // slot 0: increment a uint256 stored by the caller
    assembly {
      // load slot 0
      let v := sload(0)
      v := add(v, 1)
      sstore(0, v)
    }
  }
}
```

Recommended Mitigation:

Avoid delegatecall in sequences where the delegate target can modify shared storage in an unsafe way.

Prefer: Use regular external call (call) to isolated extension contracts that manage their own storage. If delegatecall is required (for modular wallet extension patterns), constrain and validate the delegate target (e.g., only allow trusted, pre-registered implementations), and design the called code to be safe in repeated invocations (idempotent or properly scoped). Add explicit reentrancy protections and clear state invariants before and after each delegatecall. Limit or zero out msg.value for delegatecalls unless explicitly required; document intended msg.value semantics. Concrete actions (low risk): Validate the hook/extension address against an allowlist before delegatecall. If delegatecall must be used to reuse library logic, ensure the delegated code uses only local memory/stack and does not write to critical storage slots in the caller. Add an option to use call instead of delegatecall when not required.

Add a guard that checks the implementation address is a trusted implementation:

```
+    address impl = _readHook(bytes4(msg.data));
+    require(isTrustedImplementation(impl), "untrusted delegate target")
    ;
// then delegatecall
```

Alternatively,

```
// change delegatecall to call with value 0 if storage isolation is
    required
+    (success) = LibOptim.call(call.to, 0, gasLimit == 0 ? gasleft() :
    gasLimit, call.data);
```

If delegatecall absolutely required, add comments and a registry of allowed implementations to reduce risk.

Medium

[M-1]: Unsorted Implicit Blacklist Enables Bypass.

Description:

The `_isAddressBlacklisted` function in `ImplicitSessionManager.sol` assumes the blacklist array is sorted and uses binary search. If the blacklist is not sorted, blacklisted addresses may not be detected, allowing unauthorized calls.

```
1     function _isAddressBlacklisted(address target, address[] memory
          blacklist) internal pure returns (bool) {
2    int256 left = 0;
3    int256 right = int256(blacklist.length) - 1;
4    while (left <= right) {
5      int256 mid = left + (right - left) / 2;
6      address currentAddress = blacklist[uint256(mid)];
7      if (currentAddress == target) {
8        return true;
9      } else if (currentAddress < target) {
10       left = mid + 1;
11     } else {
12       right = mid - 1;
13     }
14   }
15   return false;
16 }
```

Impact:

Blacklisted addresses can be interacted with, potentially leading to unauthorized transactions or fund loss if the blacklist is intended to prevent malicious actors.

Proof of Concept:

If blacklist = [address(3), address(1), address(2)] and target = address(3), binary search may fail to find it since it's not sorted.

Recommended Mitigation:

Ensure the blacklist is sorted before passing to the function, or implement a sorting check. Code fix: Add sorting logic or use linear search for small arrays.

```
1     // mitigation: Sort the blacklist
2     function _isAddressBlacklisted(address target, address[] memory
          blacklist) internal pure returns (bool) {
3    // Sort blacklist first (implement sort)
4    // Then proceed with binary search
5  }
```

[M-2]: abi.encodePacked() with dynamic types / hashing collision risk.

Description:

Using abi.encodePacked on dynamic-length types (bytes, string, arrays) and then passing the result to keccak256 can produce collisions because packed encoding concatenates items without padding. Different combinations of inputs may produce the same concatenation, causing hash collisions and potential authentication bypasses or integrity issues. The repo contains instances where code repeatedly appends packed data or uses abi.encodePacked for objects later hashed.

In `Permission.sol`:

```
@>      packed = abi.encodePacked(packed, ruleToPacked(...))
```

And In `Attestation.sol`:

```
@>  keccak256(toPacked(attestation)) where toPacked uses abi.
    encodePacked(...)
```

keccak256(toPacked(attestation)) where toPacked uses abi.encodePacked(...) including attestation.applicationData and bytes(authData.redirectUrl), dynamic-length fields.

Impact:

Hash collisions may allow two different attestations/permissions to produce the same digest, enabling an attacker to substitute a different message and bypass checks relying on the produced hash.

Proof of Concept:

Two different sequences of fields that under packed encoding produce identical byte sequences; e.g., abi.encodePacked(uint16(0x123), bytes1(0x456)) vs abi.encodePacked(uint8(...)) — precise examples depend on specific fields, but dynamic concatenation of bytes and uints is susceptible.

Showing collision with encodePacked combining dynamic + variable-size:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

contract PackedCollision {
  function h1(bytes memory a, bytes memory b) public pure returns (
     bytes32) {
    return keccak256(abi.encodePacked(a, b));
  }
  function h2(bytes memory ab) public pure returns (bytes32) {
    return keccak256(abi.encodePacked(ab));
  }
}
// The test is: choose a,b such that abi.encodePacked(a,b) == abi.
    encodePacked(ab) for different (a,b)
```

```
13 // Example: a = hex"12", b = hex"3456"  vs ab = hex"123456" might
       collide when boundaries are ambiguous.
```

Recommended Mitigation:

Use abi.encode (not abi.encodePacked) when producing inputs to keccak256 unless you guarantee fixed-width, non-overlapping types. For byte/ string concatenation prefer bytes.concat or explicit length prefixes.

```
1 - packed = abi.encodePacked(packed, ruleToPacked(permission.rules[i]));
2 + packed = abi.encodePacked(packed, ruleToPacked(permission.rules[i]));
      // if all elements fixed sized
3 // But safer: build an explicit encoding using abi.encode:
4 + packed = abi.encode(packed, ruleToPacked(permission.rules[i]));
```

Or better: avoid incremental encodePacked; create an array of components and then do keccak256(abi.encode(...)) for the final hash. For Attestation.toPacked used for hashing:

```
1  function toPacked(Attestation memory attestation) internal pure returns
       (bytes memory encoded) {
2    return abi.encode(
3      attestation.approvedSigner,
4      attestation.identityType,
5      attestation.issuerHash,
6      attestation.audienceHash,
7      attestation.applicationData,
8      attestation.authData.redirectUrl,
9      attestation.authData.issuedAt
10   );
11 }
12 // and use keccak256(abi.encode(...)).
```

If you need compact packed representation for storage but also need safety when hashing, hash the values individually or use abi.encode and then keccak256.

[M-3]: Incorrect Assembly Shift Parameter Order.

Description:

The report flagged uses of shl/shr with parameters potentially in swapped order; the correct built-in order is shl(shift, value) and shr(shift, value). Reported examples were:

`mstore(sub(ptr, o), shl(240, 0x3d3d))` in `Base64.sol` (line shown).

`a := and(shr(shift, word), sub(shl(mul(8, _length), 1), 1))` in `LibBytes.sol`.

`mstore(encoded, shr(152, '"challenge":"'))` in `WebAuthn.sol` and later expressions combining shr with constants.

Impact:

If these shl/shr were swapped, they'd produce incorrect values and break logic. But in this code base the instances appear correct.

Proof of Concept:

In `Base64.sol`, the line `mstore(sub(ptr, o), shl(240, 0x3d3d))` is using `shl(240, 0x3d3d)` — this means shift the value `0x3d3d` left by 240 bits. That produces `0x3d3d << 240`. Since we want to place '=' padding ascii bytes (0x3d) into the high-order bytes of a 32-byte word, `shl(240, 0x3d3d)` is correct (because 240 = 8*(32-2) to shift two bytes to the top). So the tool reported possible swap, but this instance is correct.

In `LibBytes.sol`, the expression `a := and(shr(shift, word), sub(shl(mul(8, _length), 1), 1))` is using `shr(shift, word)` and `shl(mul(8, _length), 1)` which is consistent: shift then mask. The pattern is common and correct: to extract _length bytes, shift right by shift then mask with (1 « (8*_length)) - 1. Here `shl(mul(8, _length), 1)` computes 1 « (8 * _length), which is correct. So this is fine.

In `WebAuthn.sol`, `mstore(encoded, shr(152, '"challenge":"'))` uses `shr(152, '"challenge":"')` — using shr(152, constant) shifts the constant right by 152. The code intended to temporarily store a prefix at a particular memory slot using ascii constants; treating the string constant via integer literal and shifting may be okay in assembly for packing. I verified function verify uses mstore(encoded, shr(152, '"challenge":"')) and later compares keccak256(add(o, c), q) vs keccak256(add(encoded, 0x13), q) — this matches Solady/WebAuthn patterns and is correct as written.

Conclusion: the reported "Incorrect Assembly Shift Parameter Order" warnings appear to be false positives for these instances—the code uses the assembler shift operations in the intended order and semantics.

Recommended Mitigation:

Keep current code; optionally add a tiny comment clarifying intent at non-obvious points to silence static analyzers and help future maintainers.

`Base.sol`:

```
1 // shift 0x3d3d two bytes to the top of 32-byte word (240 = 8*(32-2))
2 +  mstore(sub(ptr, o), shl(240, 0x3d3d));
```

[M-4]: Contract locks Ether without a withdraw function.

Description:

The analyzer found several contracts with payable functions (fallback/receive) but no withdraw function, concluding Ether could be locked. In this code base, several contracts (e.g., Hooks has `receive() external payable { }` and several others are full wallet modules). However, many of these are wallet modules or proxies that accept Ether intentionally, and the wallet pattern expects modules to manage funds using execute/selfExecute flows. In other words, not necessarily a bug: wallets may accept Ether and rely on owner-managed execute() to send. But if a contract is not intended to hold Ether, this is a flag.

Impact:

If the contract is not designed to withdraw Ether, funds can be locked. For wallets, the intended flow is that execute/selfExecute is used to move funds. Ensure the access control to execute is correct.

Proof of Concept:

This is the classic pattern for forwarding return data after a delegatecall. return intentionally ends the function and returns the delegated result to the caller.

// After deploying wallet and setting imageHash in storage to a value X,

// Call Estimator.estimate with _imageHash != X and signature that would be considered invalid.

// Because `Estimator._isValidImage` returns true, signatureValidation may consider image valid.

// The precise attack depends on signature flow, but the unused return is a logic bug.

Recommended Mitigation:

Audit each contract flagged to confirm whether it's intentionally able to hold Ether and whether the wallet's execute or selfExecute flow allows authorized withdrawal. If a contract should not accept Ether, remove receive()/payable from fallback or revert in fallback when there is no data. If it should accept Ether, ensure there are authorized execution paths to transfer Ether and that those paths are protected by auth checks.

```
1 - receive() external payable { }
2 + receive() external payable {
3 +   revert("receive not allowed");
4 + }
```

Or keep and ensure execute() can withdraw funds.

[M-5]: Delegatecall or callcode to an address controlled by the user.

Description:

Delegatecall where the target and function selector originate (directly or indirectly) from user-controlled data. Delegatecall runs callee code in the caller's storage context. If an attacker can control target or can influence msg.data to trigger arbitrary selectors, they might exploit storage writes, replace logic, or trick the contract into running harmful code.

Impact:

If an attacker can register an arbitrary target or the _readHook mapping can be set to an untrusted address, the attacker can deploy a malicious contract whose delegatecall will write to the wallet's storage (e.g., change owner, nonce, allowances) because delegatecall uses the caller's storage layout. Even if target is controlled only by some privileged owner, incorrect access control on hook registration or a bug in hook management could allow injection of a malicious hook.

Additionally, because msg.data (function selector + params) originates from the caller, it controls which function on target is executed. This means the attacker chooses the delegated function (by selector) and arguments, increasing the attack surface.

Proof of Concept:

Deploy a malicious contract with a function that, when delegatecalled, sets the wallet's implementation or imageHash or increments critical storage slot.

Ensure target in the wallet points to that malicious contract (this requires ability to register/replace hooks; if hook management is protected, a vulnerability may exist elsewhere).

Call the wallet with calldata selecting the malicious function (controlled by caller); the fallback will delegatecall and the malicious code will modify wallet storage.

```
1
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.18;
4
5  contract MaliciousHook {
6    // This function signature must match the selector the attacker sends
7    function attack() external {
8      // Write to storage slot 0 intentionally (for demo)
9      assembly {
10       sstore(0, 0xdeadbeef)
11     }
12   }
13 }
```

Steps: Attacker gets Hooks to point to MaliciousHook (via hook management if possible). If hook registration restricted, attacker needs to bypass that step — check hook registration logic. Call wallet

with calldata selector attack(); fallback does target.delegatecall(msg.data), attacker code executes in wallet storage context, overwriting slot 0.

Recommended Mitigation:

Enforce strict control on hook registration: Only allow trusted addresses (owner, multisig, or on-chain governance) to register hooks. Make the hook registry immutable or only modifiable via strongly-protected functions.

Validate target before delegatecall: Maintain a registry of trusted delegate implementations (mapping(address=>bool) trusted) and require trusted[target] is true.

```
+ require(trustedImplementations[target], "untrusted delegate target");
```

Restrict selectors accepted by the fallback: Validate the selector against a whitelist (e.g., only known function selectors are allowed per hook).

Add extensive tests and audits for registered hook addresses and the registration flow to ensure no unprivileged party can register malicious hooks.

```
fallback() external payable {
  if (msg.data.length >= 4) {
    bytes4 sel = bytes4(msg.data);
    address target = _readHook(sel);
+    require(target != address(0), "no hook");
+    require(trustedImplementations[target], "untrusted hook");

    // Prefer target.call if the extension should use its own storage
    (bool success, bytes memory result) = target.delegatecall(msg.data)
        ;
    assembly {
      if iszero(success) { revert(add(result, 32), mload(result)) }
      return(add(result, 32), mload(result))
    }
  }
}
```

Add trustedImplementations management functions gated by onlySelf / owner.

Confirm how hooks are registered in the repo — ensure _writeHook and addHook/removeHook are gated onlySelf . They are in `Hooks.sol`: addHook/removeHook are onlySelf). If onlySelf is secure and only the wallet can call it, risk is lower but still consider the chain of who can call onlySelf (self-execute can be called only by the contract itself; check modules that can call it). If hook registry is only writable by the wallet itself via internal governance, document and keep strong tests.

For more detailed read this:Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

Low

[L-1]: Shadow Functions Selector on Hooks Contract.

Description:

The `Hooks.sol` contract's fallback function intercepts selectors for standard functions like receive(), onERC721Received(), etc., preventing users from hooking these functions. Code: The fallback in Hooks.sol catches all calls.

Impact:

Users cannot customize behavior for token receives or ETH transfers via hooks.

Proof of Concept:

Attempting to hook receive will be intercepted by the fallback instead of the hook.

Recommendation Mitigation:

Move hook management to a separate contract without fallback, or exclude certain selectors from fallback. Code: Refactor Hooks.sol to delegate specific selectors.

Informational

[I-6]: `ecrecover` is susceptible to signature malleability

The `ecrecover` function is susceptible to signature malleability. This means that the same message can be signed in multiple ways, allowing an attacker to change the message signature without invalidating it. This can lead to unexpected behavior in smart contracts, such as the loss of funds or the ability to bypass access control. Consider using OpenZeppelin's ECDSA library instead of the built-in function.

5 Found Instances

- Found in src/extensions/recovery/Recovery.sol Line: 203

```
1         address addr = ecrecover(rPayloadHash, v, r, s);
```

- Found in src/extensions/sessions/SessionSig.sol Line: 116

```
1             address recoveredIdentitySigner = ecrecover(
                attestationHash, v, r, s);
```

- Found in src/extensions/sessions/SessionSig.sol Line: 170

```
callSignature.sessionSigner = ecrecover(callHash, v, r,
    s);
```

- Found in src/modules/auth/BaseSig.sol Line: 233

```
address addr = ecrecover(_opHash, v, r, s);
```

- Found in src/modules/auth/BaseSig.sol Line: 398

```
address addr = ecrecover(keccak256(abi.encodePacked("\
    x19Ethereum Signed Message:\n32", _opHash)), v, r, s)
    ;
```

[I-2]: Solidity pragma should be specific, not wide

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.0;`, use `pragma solidity 0.8.0;`

49 Found Instances

- Found in src/Estimator.sol Line: 2

```
pragma solidity ^0.8.27;
```

- Found in src/Factory.sol Line: 2

```
pragma solidity ^0.8.27;
```

- Found in src/Guest.sol Line: 2

```
pragma solidity ^0.8.27;
```

- Found in src/Simulator.sol Line: 2

```
pragma solidity ^0.8.27;
```

- Found in src/Stage1Module.sol Line: 2

```
pragma solidity ^0.8.27;
```

- Found in src/Stage2Module.sol Line: 2

```
pragma solidity ^0.8.27;
```

- Found in src/Wallet.sol Line: 2

```
pragma solidity ^0.8.0;
```

- Found in src/extensions/passkeys/Passkeys.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/recovery/Recovery.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/SessionErrors.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/SessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/SessionSig.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/ExplicitSessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/IExplicitSessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/Permission.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/PermissionValidator.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/implicit/Attestation.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/implicit/ISignalsImplicitMode.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/implicit/ImplicitSessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Calls.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/ERC4337v07.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/Hooks.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Implementation.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Nonce.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Payload.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/ReentrancyGuard.sol Line: 2

```
1 pragma solidity ^0.8.0;
```

- Found in src/modules/Storage.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/BaseAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/BaseSig.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/SelfAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/Stage1Auth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/Stage2Auth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/IAccount.sol Line: 2

```
1 pragma solidity ^0.8.28;
```

- Found in src/modules/interfaces/IAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/ICheckpointer.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/IDelegatedExtension.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/IERC1155Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC1271.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC223Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC721Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC777Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IEntryPoint.sol Line: 2

```
1 pragma solidity ^0.8.28;
```

- Found in src/modules/interfaces/IPartialAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/ISapient.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/utils/Base64.sol Line: 2

```
1 pragma solidity ^0.8.4;
```

- Found in src/utils/LibBytes.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/utils/LibOptim.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/utils/P256.sol Line: 2

```
1 pragma solidity ^0.8.4;
```

- Found in src/utils/WebAuthn.sol Line: 2

```
1 pragma solidity ^0.8.4;
```

[I-3]: `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

2 Found Instances

- Found in src/Factory.sol Line: 18

```
1   function deploy(address _mainModule, bytes32 _salt) public
       payable returns (address _contract) {
```

- Found in src/extensions/recovery/Recovery.sol Line: 64

```
1   function totalQueuedPayloads(address _wallet, address _signer)
       public view returns (uint256) {
```

[I-4]: Define and use `constant` variables instead of using literals.

If the same constant literal value is used multiple times, create a constant state variable and reference it throughout the contract.

45 Found Instances

- Found in src/extensions/sessions/SessionSig.sol Line: 263

```
1           uint256 sizeSize = uint8(firstByte & 0x0f);
```

- Found in src/extensions/sessions/SessionSig.sol Line: 311

```
        uint256 blacklistCount = uint256(firstByte & 0x0f);
```

- Found in src/extensions/sessions/SessionSig.sol Line: 312

```
        if (blacklistCount == 0x0f) {
```

- Found in src/modules/Payload.sol Line: 142

```
    if (globalFlag & 0x01 == 0x01) {
```

- Found in src/modules/Payload.sol Line: 159

```
    if (globalFlag & 0x10 == 0x10) {
```

- Found in src/modules/Payload.sol Line: 163

```
      if (globalFlag & 0x20 == 0x20) {
```

- Found in src/modules/Payload.sol Line: 179

```
      if (flags & 0x01 == 0x01) {
```

- Found in src/modules/Payload.sol Line: 188

```
      if (flags & 0x02 == 0x02) {
```

- Found in src/modules/Payload.sol Line: 193

```
      if (flags & 0x04 == 0x04) {
```

- Found in src/modules/Payload.sol Line: 202

```
      if (flags & 0x08 == 0x08) {
```

- Found in src/modules/Payload.sol Line: 207

```
      _decoded.calls[i].delegateCall = (flags & 0x10 == 0x10);
```

- Found in src/modules/Payload.sol Line: 210

```
      _decoded.calls[i].onlyFallback = (flags & 0x20 == 0x20);
```

- Found in src/modules/auth/BaseAuth.sol Line: 43

```
    return (address(uint160(word >> 96)), uint256(uint96(word)));
```

- Found in src/modules/auth/BaseAuth.sol Line: 48

```
        STATIC_SIGNATURE_KEY, _hash, bytes32(uint256(uint160(
            _address)) << 96 | (_timestamp & 0
            xffffffffffffffffffffffff))
```

- Found in src/modules/auth/BaseAuth.sol Line: 88

```
    if (signatureFlag & 0x80 == 0x80) {
```

- Found in src/modules/auth/BaseSig.sol Line: 88

```
    if (signatureFlag & 0x40 == 0x40 && _checkpointer == address
        (0)) {
```

- Found in src/modules/auth/BaseSig.sol Line: 109

```
    if (signatureFlag & 0x01 == 0x01) {
```

- Found in src/modules/auth/BaseSig.sol Line: 114

```
    _payload.noChainId = signatureFlag & 0x02 == 0x02;
```

- Found in src/modules/auth/BaseSig.sol Line: 223

```
            uint8 addrWeight = uint8(firstByte & 0x0f);
```

- Found in src/modules/auth/BaseSig.sol Line: 247

```
            uint8 addrWeight = uint8(firstByte & 0x0f);
```

- Found in src/modules/auth/BaseSig.sol Line: 269

```
            uint8 addrWeight = uint8(firstByte & 0x03);
```

- Found in src/modules/auth/BaseSig.sol Line: 279

```
            uint256 sizeSize = uint8(firstByte & 0x0c) >> 2;
```

- Found in src/modules/auth/BaseSig.sol Line: 315

```
            uint256 sizeSize = uint8(firstByte & 0x0f);
```

- Found in src/modules/auth/BaseSig.sol Line: 338

```
            uint256 externalWeight = uint8(firstByte & 0x0c) >> 2;
```

- Found in src/modules/auth/BaseSig.sol Line: 343

```
            uint256 internalThreshold = uint8(firstByte & 0x03);
```

- Found in src/modules/auth/BaseSig.sol Line: 388

```
            uint8 addrWeight = uint8(firstByte & 0x0f);
```

- Found in src/modules/auth/BaseSig.sol Line: 432

```
            uint8 addrWeight = uint8(firstByte & 0x03);
```

- Found in src/modules/auth/BaseSig.sol Line: 443

```
              uint256 sizeSize = uint8(firstByte & 0x0c) >> 2;
```

- Found in src/modules/auth/BaseSig.sol Line: 468

```
            uint8 addrWeight = uint8(firstByte & 0x03);
```

- Found in src/modules/auth/BaseSig.sol Line: 477

```
            uint256 sizeSize = uint8(firstByte & 0x0c) >> 2;
```

- Found in src/utils/LibBytes.sol Line: 115

```
      uint256 yParity = uint256(yParityAndS >> 255);
```

- Found in src/utils/LibBytes.sol Line: 116

```
      s = bytes32(uint256(yParityAndS) & ((1 << 255) - 1));
```

[I-5]: Event is missing `indexed` fields.

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

12 Found Instances

- Found in src/extensions/recovery/Recovery.sol Line: 29

```
    event NewQueuedPayload(address _wallet, address _signer, bytes32
        _payloadHash, uint256 _timestamp);
```

- Found in src/extensions/sessions/explicit/PermissionValidator.sol Line: 16

```
    event LimitUsageUpdated(address wallet, bytes32 usageHash,
       uint256 usageAmount);
```

- Found in src/modules/Calls.sol Line: 18

```
1    event CallSucceeded(bytes32 _opHash, uint256 _index);
```

- Found in src/modules/Calls.sol Line: 20

```
1    event CallFailed(bytes32 _opHash, uint256 _index, bytes
        _returnData);
```

- Found in src/modules/Calls.sol Line: 22

```
1    event CallAborted(bytes32 _opHash, uint256 _index, bytes
        _returnData);
```

- Found in src/modules/Calls.sol Line: 24

```
1    event CallSkipped(bytes32 _opHash, uint256 _index);
```

- Found in src/modules/Hooks.sol Line: 20

```
1    event DefinedHook(bytes4 selector, address implementation);
```

- Found in src/modules/Implementation.sol Line: 12

```
1    event ImplementationUpdated(address newImplementation);
```

- Found in src/modules/Nonce.sol Line: 12

```
1    event NonceChange(uint256 _space, uint256 _newNonce);
```

- Found in src/modules/auth/BaseAuth.sol Line: 37

```
1    event StaticSignatureSet(bytes32 _hash, address _address, uint96
        _timestamp);
```

- Found in src/modules/auth/Stage1Auth.sol Line: 30

```
1    event ImageHashUpdated(bytes32 newImageHash);
```

- Found in src/modules/auth/Stage2Auth.sol Line: 18

```
1    event ImageHashUpdated(bytes32 newImageHash);
```

[I-6]: PUSH0 is not supported by all chains.

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in

case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

49 Found Instances

- Found in src/Estimator.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/Factory.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/Guest.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/Simulator.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/Stage1Module.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/Stage2Module.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/Wallet.sol Line: 2

```
1 pragma solidity ^0.8.0;
```

- Found in src/extensions/passkeys/Passkeys.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/recovery/Recovery.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/SessionErrors.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/SessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/SessionSig.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/ExplicitSessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/IExplicitSessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/Permission.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/explicit/PermissionValidator.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/implicit/Attestation.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/implicit/ISignalsImplicitMode.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/extensions/sessions/implicit/ImplicitSessionManager.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Calls.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/ERC4337v07.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/Hooks.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Implementation.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Nonce.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/Payload.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/ReentrancyGuard.sol Line: 2

```
1 pragma solidity ^0.8.0;
```

- Found in src/modules/Storage.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/BaseAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/BaseSig.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/SelfAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/Stage1Auth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/auth/Stage2Auth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/IAccount.sol Line: 2

```
1 pragma solidity ^0.8.28;
```

- Found in src/modules/interfaces/IAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/ICheckpointer.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/IDelegatedExtension.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/IERC1155Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC1271.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC223Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC721Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IERC777Receiver.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/modules/interfaces/IEntryPoint.sol Line: 2

```
1 pragma solidity ^0.8.28;
```

- Found in src/modules/interfaces/IPartialAuth.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/modules/interfaces/ISapient.sol Line: 2

```
1 pragma solidity ^0.8.27;
```

- Found in src/utils/Base64.sol Line: 2

```
1 pragma solidity ^0.8.4;
```

- Found in src/utils/LibBytes.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/utils/LibOptim.sol Line: 2

```
1 pragma solidity ^0.8.18;
```

- Found in src/utils/P256.sol Line: 2

```
1 pragma solidity ^0.8.4;
```

- Found in src/utils/WebAuthn.sol Line: 2

```
1 pragma solidity ^0.8.4;
```

L-7: Empty Block

Consider removing empty blocks.

1 Found Instances

- Found in src/modules/Hooks.sol Line: 87

```
1    function tokensReceived(
```

L-8: Internal functions called only once can be inlined

Instead of separating the logic into a separate function, consider inlining the logic into the calling function. This can reduce the number of function calls and improve readability.

Gas