CANTINA

Clave

Security Review

Cantina Managed review by:
Riley Holterhus, Lead Security Researcher
Blockdev, Security Researcher

December 15, 2023

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
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.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Clave is an easy-to-use non-custodial smart wallet powered by Account Abstraction and the Hardware Elements (e.g Secure Enclave, Android Trustzone etc.), offering a unique onboarding process.

From Nov 1st to Nov 15th the Cantina team conducted a review of clave-monorepo on commit hash bbaabc87. The team identified a total of **24** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 3
- Medium Risk: 4
- Low Risk: 4
- Gas Optimizations: 3
- Informational: 10

3 Findings

3.1 High Risk

3.1.1 `isValidSignature()` signature replay in accounts with shared owners

Severity: High Risk

Context: ERC1271Handler.sol

Description: To validate a signature using the ERC1271 standard, the Clave `isValidSignature()` utilizes its `_handleValidation()` function on the `signedHash`:

```
function isValidSignature(
    bytes32 signedHash,
    bytes calldata signatureAndValidator
) external view override returns (bytes4 magicValue) {
    (bytes memory signature, address validator) = SignatureDecoder.decodeSignatureNoHookData(
        signatureAndValidator
    );

    bool valid = _handleValidation(validator, signedHash, signature);

    magicValue = valid ? _ERC1271_MAGIC : bytes4(0);
}
```

Since `_handleValidation()` directly forwards the `signedHash` to the `validator`, this implementation only checks that the raw `signedHash` has been signed by one of the k1/r1 owners of the account. If multiple Clave accounts share an owner, one signature can be valid for all accounts, which could allow for signature replay.

Recommendation: To ensure that a signature is only valid for a specific Clave account, the ERC1271 `signedHash` argument can be additionally hashed with `address(this)` before forwarding to the `_handleValidation()` function. To do this in a standard way, consider using EIP712, which includes `address(this)` as the `verifyingContract` in the domain separator. This also allows you to define a custom EIP712 typestruct (such as "`ClaveMessage(bytes message)`") to use when signing an ERC1271 message.

Additionally, note that the `EOAValidator` *does* convert the `signedHash` using `toEthSignedMessageHash()`, which is a conversion that's useful for differentiating between signed messages and signed RLP encoded transactions. Since EIP712 also makes this differentiation, the `toEthSignedMessageHash()` conversion is not strictly necessary in the `isValidSignature()` scenario. So, as part of the overall change, consider only applying `toEthSignedMessageHash()` to the `signedHash` in the `_validateTransaction()` path.

Clave: Fixed in commit 8f29439f. Additionally removed `toEthSignedMessageHash()` in commit 68b8678d, since all AA transactions are signed using EIP-712 which makes them distinct from RLP encoded transactions already.

Cantina Managed: Verified.

3.1.2 `clear()` only deletes first element from linked list

Severity: High Risk

Context: LinkedList.sol#L121-L130, LinkedList.sol#L292-L301

Description: The linked list `clear()` functions have the following definition (using the `AddressLinkedList` as an example):

```
function clear(mapping(address => address) storage self) internal {
    for (
        address cursor = self[SENTINEL_ADDRESS];
        uint160(cursor) > SENTINEL_UINT;
        cursor = self[cursor]
    ) {
        delete self[cursor];
    }
    delete self[SENTINEL_ADDRESS];
}
```

In this code, notice that the `delete self[cursor]` statement will happen immediately before the `cursor = self[cursor]` advancement in the `for` loop. Since `self[cursor]` is being deleted before advancing, the loop will always exit after the first iteration, and the list won't be cleared as expected.

As a consequence, the `resetOwners()` function (used by recovery modules) will not correctly clear the previous owners.

Recommendation: To properly delete all elements, use a temporary `nextCursor` value as follows:

```
function clear(mapping(address => address) storage self) internal {
    address nextCursor;
    for (
        address cursor = self[SENTINEL_ADDRESS];
        uint160(cursor) > SENTINEL_UINT;
        cursor = nextCursor
    ) {
        nextCursor = self[cursor];
        delete self[cursor];
    }
    delete self[SENTINEL_ADDRESS];
}
```

Alternatively, consider using a `do-while` loop in a similar way:

```
function clear(mapping(address => address) storage self) internal {
    address cursor = SENTINEL_ADDRESS;
    do {
        address nextCursor = self[cursor];
        delete self[cursor];
        cursor = nextCursor;
    } while (uint160(cursor) > SENTINEL_UINT);
}
```

Clave: Fixed in commit 2bba7a5e.

Cantina Managed: Verified.

3.1.3 `isEmpty()` does not account for sentinel pointing to itself

Severity: High Risk

Context: LinkedList.sol#L152, LinkedList.sol#L323

Description: An empty linked list can have the sentinel element pointing to itself if all elements are removed using `remove()`. In this case, `self[SENTINEL_BYTES] = SENTINEL_BYTES` for bytes linked list, and `self[SENTINEL_ADDRESS] = SENTINEL_ADDRESS`. However, `isEmpty()` doesn't account for this case, and will incorrectly consider the list as non-empty.

As a consequence, the `_r1RemoveValidator()` and `_r1RemoveOwner()` functions will not correctly prevent users from bricking their accounts by removing all validators/owners.

Recommendation:

- Update `BytesLinkedList.isEmpty()` as follows:

```
- return self[SENTINEL_BYTES].length == 0;
+ return self[SENTINEL_BYTES].length <= SENTINEL_LENGTH;
```

- Update `AddressLinkedList.isEmpty()` as follows:

```
- return self[SENTINEL_ADDRESS] == address(0);
+ return self[SENTINEL_ADDRESS] <= SENTINEL_ADDRESS;
```

Clave: Fixed with commit 2bba7a5e.

Cantina Managed: Verified.

3.2 Medium Risk

3.2.1 CloudRecoveryModule never increments the recovery nonce

Severity: Medium Risk

Context: CloudRecoveryModule.sol#L96-L128

Description: When the `startRecovery()` function is called on a recovery module, it is intended that the account's recovery `nonce` is incremented. However, this is currently missing from the `CloudRecoveryModule`.

As a consequence, anyone can replay an account's cloud recovery. This could temporarily disable additional recovery attempts, and if the private key for the original recovery's `newOwner` has been lost, would lock the account.

Recommendation: Increment the `recoveryNonces` mapping in `CloudRecoveryModule`:

```
 function startRecovery(RecoveryData calldata recoveryData, bytes calldata signature) external {
     address recoveringAddress = recoveryData.recoveringAddress;

     if (recoveryData.nonce != recoveryNonces[recoveringAddress]) {
         revert Errors.INVALID_RECOVERY_NONCE();
     }

     if (isRecovering(recoveringAddress)) {
         revert Errors.RECOVERY_IN_PROGRESS();
     }

     if (!isInited(recoveringAddress)) {
         revert Errors.RECOVERY_NOT_INITED();
     }

     bytes32 eip712Hash = _hashTypedDataV4(_recoveryDataHash(recoveryData));
     address guardian = cloudGuardian[recoveringAddress];

     if (!guardian.isValidSignatureNow(eip712Hash, signature)) {
         revert Errors.INVALID_GUARDIAN_SIGNATURE();
     }

     recoveryStates[recoveryData.recoveringAddress] = RecoveryState({
         timelockExpiry: block.timestamp + TIMELOCK,
         newOwner: recoveryData.newOwner
     });

+    recoveryNonces[recoveringAddress]++;

     emit RecoveryStarted(
         recoveryData.recoveringAddress,
         recoveryData.newOwner,
         block.timestamp + TIMELOCK
     );
 }
```

Clave: Fixed with commit 989bc989.

Cantina Managed: Verified.

3.2.2 Disallow changing cloud guardian during recovery

Severity: Medium Risk

Context: CloudRecoveryModule.sol#L74-L87

Description: An account can change its cloud guardian address by calling the `updateGuardian()` function on the `CloudRecoveryModule`. The comments of this function state that "Recovery must not be in progress for the account", but this is currently not enforced.

Recommendation: Disallow changing the cloud guardian address if a recovery is in progress for the account:

```
  /**
   * @notice Update the guardian for the calling account
   * @dev Recovery must not be in progress for the account
   * @dev Module must be inited for the account
   * @dev Guardian must not be the zero address
   * @param guardian Address of the new guardian
   */
  function updateGuardian(address guardian) external {
      if (!isInited(msg.sender)) {
          revert Errors.RECOVERY_NOT_INITED();
      }

+     if (isRecovering(msg.sender)) {
+         revert Errors.RECOVERY_IN_PROGRESS();
+     }

      _updateGuardian(guardian);
  }
```

Clave: Fixed with commit e6255fb4.

Cantina Managed: Verified.

3.2.3 Use OpenZeppelin's SafeERC20

Severity: Medium Risk

Context: ERC20Paymaster.sol#L113, ERC20Paymaster.sol#L200

Description: Some ERC20 tokens may not follow the entire ERC20 specification. For example, `transfer()` and `transferFrom()` are expected to return `true` and revert on any failure, but USDT doesn't return any value. OpenZeppelin `SafeERC20` library handles these cases.

Recommendation: Consider using OpenZeppelin's `SafeERC20`'s `safeTransfer()` and `safeTransferFrom()` functions instead of calling `transfer()` and `transferFrom()` on the token directly.

Clave: Fixed with PR 716.

Cantina Managed: Verified.

3.2.4 Malicious module/hook may not be removed

Severity: Medium Risk

Context: ModuleManager.sol#L111-L112, HookManager.sol#L197-L198

Description: In the `ModuleManager`, `_removeModule()` calls the module's `disable()` function using a low-level call, allowing the overall transaction to succeed even if the module reverts. Similar logic would be expected in the `HookManager` function `_removeHook()`, but this is currently not the case:

```
//TODO: turn into low level call
IHook(hook).disable();
```

Moreover, even when a low-level call is utilized, there exist theoretical "gas-griefing" tactics where a module or hook can block its removal. Firstly, if the low-level call doesn't set a specific `gas` limit, it forwards 63/64 of the remaining gas. If the contract deliberately uses up this entire amount, the outer transaction may experience an out-of-gas error. Secondly, if the low-level call returns a large amount of data, it can force the account to incur memory expansion costs, which can also cause out-of-gas errors.

Recommendation: Firstly, as noted by the TODO comment, change `_removeHook()` to use a low-level call where the success of `disable()` is ignored.

Secondly, to prevent potential "gas-griefing" issues, consider using an explicit `gas` amount in these low-level calls, and limit the memory expansion from the call by using assembly or a library like ExcessivelySafeCall.

Clave: Fixed in PR 773.

Cantina: The fix uses `gasleft()` as input to `excessivelySafeCall()`, and technically speaking, this means a module/hook could waste up to 63/64 of the remaining gas when it gets control flow in `disable()` (note:

this 63/64 number is coming from EIP-150). If a module/hook were to do this, you *could* just increase the tx's gas amount until 1/64 is sufficient for the whole tx to complete - but this would be more expensive and would also need to consider block gas limits.

So, to be extra safe, we would recommend adding a `uint256 gas` argument to `removeModule()` and `removeHook()`, and use that value instead of `gasleft()`.

Clave: We are okay with the gas griefing probability as we will be developing the modules/hooks.

Cantina Managed: Verified.

3.3 Low Risk

3.3.1 Unbounded linked list traversals

Severity: Low Risk

Context: LinkedList.sol, HookManager.sol, ValidationHandler.sol

Description: The Clave account implementation uses linked lists in various locations, and in some places, these lists are fully traversed. Since there is no maximum size, these traversals can become arbitrarily expensive in terms of gas. In an extreme scenario, a full list traversal will cost more than the block gas limit, and will make some functions impossible to call.

Recommendation: Consider enforcing a maximum size on the account's linked lists. Alternatively, since out-of-gas issues are very unlikely to happen in normal circumstances, consider simply documenting this behavior as a warning in the front end/comments.

Clave: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 `CLAVE_STORAGE_SLOT` should be subtracted by one

Severity: Low Risk

Context: ClaveStorage.sol#L5

Description: The `ClaveStorage` library maintains the contract's `Layout` struct in the storage slot equal to `keccak256('clave.contracts.ClaveStorage')`. This is similar to how EIP-1967 storage slots are maintained, except that there is an offset of `-1` missing in the calculation.

The offset is recommended because the resulting value wouldn't have a known preimage, which decreases the chance of a collision with a compiler storage slot.

Recommendation: Subtract one from the `CLAVE_STORAGE_SLOT`:

```
- bytes32 private constant CLAVE_STORAGE_SLOT = keccak256('clave.contracts.ClaveStorage');
+ bytes32 private constant CLAVE_STORAGE_SLOT = bytes32(uint256(keccak256('clave.contracts.ClaveStorage')) -
1);
```

Clave: Fixed with commit da6b75a3.

Cantina Managed: Verified.

3.3.3 Unsafe `address` casting

Severity: Low Risk

Context: ClaveImplementation.sol#L218

Description: In the zkSync `Transaction` struct, the `from`, `to`, and `paymaster` fields are defined as `uint256` values, even though they each represent an `address` value (which is equivalent to `uint160` under-the-hood). As a result, it is possible for these values to overflow if the conversion is not carefully handled. For zkSync's native transactions, this is not a concern, since the bootloader enforces that no overflow happens.

However, an alternative usage of the zkSync `Transaction` struct exists in the `executeTransactionFromOutside()` function. This function will be called in a transaction originating by another address, so the bootloader will not have inspected any of the values for overflow. So, when this code eventually runs:

```
function _executeTransaction(
    Transaction calldata transaction
) internal runExecutionHooks(transaction) {
    address to = address(uint160(transaction.to));
    uint128 value = Utils.safeCastToU128(transaction.value);
    bytes calldata data = transaction.data;
    uint32 gas = Utils.safeCastToU32(gasleft());

    // ... code omitted ...
}
```

the `address(uint160(transaction.to))` casting can silently overflow. Fortunately, since the full `to` value would have been signed by the user, there isn't any direct way to exploit this. However, it does allow odd behavior and would make more sense to be disallowed.

Recommendation: When casting any of the `Transaction` struct values to an address, ensure that no overflow happens:

```diff
+ function _safeCastToAddress(uint256 value) internal returns (address) {
+     if (value > type(uint160).max) revert();
+     return address(uint160(value));
+ }

  function _executeTransaction(
      Transaction calldata transaction
  ) internal runExecutionHooks(transaction) {
-     address to = address(uint160(transaction.to));
+     address to = _safeCastToAddress(transaction.to);
      uint128 value = Utils.safeCastToU128(transaction.value);
      bytes calldata data = transaction.data;
      uint32 gas = Utils.safeCastToU32(gasleft());

      // ... code omitted ...
  }
```

Clave: Fixed in commit 4ca01555.

Cantina Managed: Verified.

3.3.4 Disallow account directly calling module's `init()` and `disable()` functions

Severity: Low Risk

Context: CloudRecoveryModule.sol#L41-L72, SocialRecoveryModule.sol#L55-L86

Description: If a user wants to add a module to their Clave account, they are supposed to do a self-call to invoke their `addModule()` function. This function adds the module to an internal linked list and then calls the `init()` function on the module.

Technically, there is nothing stopping a user from calling the `init()` function directly on the module. Even though the module would accept this call and update its own storage, there would be issues later on, since the module will never be added to the account's linked list. This might be done by accident, especially since most module functions *are* intended to be called directly (e.g. `updateGuardian()`, `updateConfig()`, and `stopRecovery()`). The same potential problem exists with `removeModule()` and the `disable()` function.

Recommendation: Consider explicitly preventing this user error. One way to do this would be making the following changes to the module's functions (using the `SocialRecoveryModule` as an example):

```
  function init(bytes calldata initData) external override {
      if (isInited(msg.sender)) {
          revert Errors.ALREADY_INITED();
      }

+     if (!IClaveAccount(msg.sender).isModule(address(this)) {
+         revert Errors.MODULE_NOT_ADDED_CORRECTLY();
+     }

      RecoveryConfig memory config = abi.decode(initData, (RecoveryConfig));


      emit Inited(msg.sender);

      _updateConfig(config);
  }

  function disable() external override {
      if (!isInited(msg.sender)) {
          revert Errors.RECOVERY_NOT_INITED();
      }

+     if (IClaveAccount(msg.sender).isModule(address(this)) {
+         revert Errors.MODULE_NOT_REMOVED_CORRECTLY();
+     }

      delete recoveryConfigs[msg.sender];

      emit Disabled(msg.sender);

      _stopRecovery();
  }
```

This code checks that the module is appropriately present/absent from the account. This works because the `ModuleManager` updates the linked lists *before* calling `init()` or `disable()`.

Clave: Fixed with commit 328614dc.

Cantina Managed: Verified.

3.4 Gas Optimization

3.4.1 `EOAValidator.validateSignature()` can be simplified

Severity: Gas Optimization

Context: EOAValidator.sol#L25

Description: OpenZeppelin's `ECDSA.tryRecover()` reads as follows:

```
address signer = ecrecover(hash, v, r, s);
if (signer == address(0)) {
    return (address(0), RecoverError.InvalidSignature);
}

return (signer, RecoverError.NoError);
```

`EOAValidator.validateSignature()` calls `tryRecover()` and again checks the error to return `address(0)` if an error is returned:

```
signer = recoverError == ECDSA.RecoverError.NoError ? recoveredAddress : address(0);
```

However, this check is redundant: if `recoverError != ECDSA.RecoverError.NoError`, `tryRecover()` always returns `address(0)` for `recoveredAddress`.

Recommendation: Update EOAValidator.sol#L21-L25 as follows:

```
- (address recoveredAddress, ECDSA.RecoverError recoverError) = signedHash
+ (signer, ECDSA.RecoverError recoverError) = signedHash
      .toEthSignedMessageHash()
      .tryRecover(signature);

- signer = recoverError == ECDSA.RecoverError.NoError ? recoveredAddress : address(0);
```

Clave: Fixed with commits d5297348 and 002f5d71.

Cantina Managed: Verified.

3.4.2 `validateSignature()` can be simplified

Severity: Gas Optimization

Context: TEEValidator.sol#L28

Description: This `if` clause can be simplified as `valid` is true iff `callVerifier(signedHash, rs, pubKey)` is true;

```
if (callVerifier(signedHash, rs, pubKey)) valid = true;
```

Recommendation: Update TEEValidator.sol#L28 as follows:

```
-if (callVerifier(signedHash, rs, pubKey)) valid = true;
+valid = callVerifier(signedHash, rs, pubKey);
```

Clave: Fixed with commit a3aa0985.

Cantina Managed: Verified.

3.4.3 Store validation and execution hooks in different linked lists

Severity: Gas Optimization

Context: HookManager.sol#L108-L109

Description: A hook can be a validation hook, an execution hook or both. `runValidationHooks()` and `runExecutionHooks()` iterate through a linked list storing hooks to find validation and execution hooks respectively. Storing validation and execution hooks in different lists will save gas for `runValidationHooks()` and `runExecutionHooks()`.

Recommendation: Consider storing validation and execution hooks in different lists. If you decide to implement this change, be careful of the case where a hook is both a validation and execution hook. In this case, that hook has to be added to both lists.

Clave: Fixed in PR 787.

Cantina Managed: Verified.

3.5 Informational

3.5.1 Document hook address mining

Severity: Informational

Context: HookManager.sol#L241-L244

Description: The `HookManager` contract classifies hooks as validation hooks, execution hooks or both. This classification is based on two bitmasks applied to the hook's address, which implies that hook addresses must be mined before deployment to find appropriate values for these bits. However, this process isn't documented in the code.

Recommendation: Document this behavior in the comments of the `HookManager`.

Clave: Fixed with PR 787.

Cantina Managed: Requirement on hook address is removed with this fix as it now stores validation and execution hooks in different lists. Verified.

3.5.2 Always use `delete` when resetting values in linked lists

Severity: Informational

Context: LinkedList.sol#L59, LinkedList.sol#L230

Description: In the two linked list libraries, the `replace()` function must erase an existing value from the linked list. For the `BytesLinkedList`, this is done by setting `self[_value] = bytes('')`, and for the `AddressLinkedList`, this is done by setting `self[_value] = address(0)`.

In both cases, the same logic can be accomplished by instead calling `delete self[_value]`, which is more consistent with the rest of the codebase.

Recommendation: Instead of manually setting values to their defaults, use the `delete` keyword.

Clave: Fixed with 2bba7a5e.

Cantina Managed: Verified.

3.5.3 Invalid signature reverts instead of returning `bytes4(0)`

Severity: Informational

Context: ValidationHandler.sol#L36-L43

Description: In native zkSync account abstraction, it is intended that the `validateTransaction()` function returns `bytes4(0)` if validation fails. As part of this, the Clave account's `_handleValidation()` function returns a boolean to indicate if validation succeeds or not. For secp256k1 signature validation specifically, this code is used:

```
address recoveredAddress = IK1Validator(validator).validateSignature(
    signedHash,
    signature
);

if (OwnerManager._k1IsOwner(recoveredAddress)) {
    return true;
}
```

In the current EOAValidator implementation, an invalid signature length will lead to the `recoveredAddress` being returned as `address(0)`. On the other hand, the `_k1IsOwner()` function reverts if the argument is `address(0)`, since this is not a valid address in the `AddressLinkedList` library.

So, invalid signatures will incorrectly revert the `validateTransaction()` call instead of causing a `bytes4(0)` return value. Since this deviates from the intended zkSync behavior, the error might not be correctly handled off-chain.

Recommendation: Remove this revert by either changing the `_k1IsOwner()` behavior or by exiting early as follows:

```
  address recoveredAddress = IK1Validator(validator).validateSignature(
      signedHash,
      signature
  );

+ if (recoveredAddress == address(0)) {
+     return false;
+ }

  if (OwnerManager._k1IsOwner(recoveredAddress)) {
      return true;
  }
```

Clave: Fixed with commit bcd0a6d9.

Cantina Managed: Verified.

3.5.4 `updateConfig()` can be simplified

Severity: Informational

Context: SocialRecoveryModule.sol#L94-L104

Description: The `SocialRecoveryModule` contains the following function:

```
function updateConfig(RecoveryConfig memory config) external {
    if (recoveryConfigs[msg.sender].timelock == 0) {
        revert Errors.RECOVERY_NOT_INITED();
    }

    if (recoveryStates[msg.sender].timelockExpiry != 0) {
        revert Errors.RECOVERY_IN_PROGRESS();
    }

    _updateConfig(config);
}
```

Also, this contract has two helper functions with the following logic:

```
function isInited(address account) public view override returns (bool) {
    return recoveryConfigs[account].timelock != 0;
}

function isRecovering(address account) public view returns (bool) {
    return recoveryStates[account].timelockExpiry != 0;
}
```

Recommendation: Since these helper functions contain similar logic to the `updateConfig()` code, consider utilizing them there:

```
function updateConfig(RecoveryConfig memory config) external {
    if (!isInited(msg.sender)) {
        revert Errors.RECOVERY_NOT_INITED();
    }

    if (isRecovering(msg.sender)) {
        revert Errors.RECOVERY_IN_PROGRESS();
    }

    _updateConfig(config);
}
```

Clave: Fixed with commit 127e4d5e.

Cantina Managed: Verified.

3.5.5 Implement separate interfaces for validation hooks and execution hooks

Severity: Informational

Context: IHook.sol

Description: Clave accounts are compatible with two types of hooks: validation hooks and execution hooks. Both of these hooks use the same `IHook` interface, which means both hooks must implement functions that they may not actually use. Pure validation hooks will need to implement dummy `preExecutionHook()`/`postExecutionHook()` functions, while pure execution hooks will need to implement a dummy `validationHook()` function.

Recommendation: Consider simplifying hook development by splitting these functions into two separate interfaces: `IValidationHook.sol` and `IExecutionHook.sol`.

Clave: Fixed with PR 787.

Cantina Managed: Verified.

3.5.6 Redundant `uint160` casting for `address` comparisons

Severity: Informational

Context: LinkedList.sol, HookManager.sol, SocialRecoveryModule.sol

Description: Throughout the codebase, there are a few instances of two `address` values being cast to `uint160` for the purpose of comparing their values. Since it is possible to compare `address` values directly in Solidity, these castings are technically redundant.

Recommendation: Consider simplifying the codebase by removing these redundant `uint160` casts.

Clave: Fixed with commits 83c114a5 and 2bba7a5e.

Cantina Managed: Verified.

3.5.7 Linked list iteration can be simplified

Severity: Informational

Context: LinkedList.sol

Description: In the two linked list libraries, the `replace()`, `remove()`, `clear()`, `size()` and `list()` functions traverse the linked list using a `cursor` that is initialized to `self[SENTINEL_VALUE]` (where `SENTINEL_VALUE` is either `SENTINEL_BYTES` or `SENTINEL_ADDRESS`, depending on which version you are using). In the particular case of `replace()` and `remove()`, the main loop checks whether `self[cursor]` equals the value of interest, and if so, it updates the linked list. This means that the first loop iteration will inspect `self[self[SENTINEL_VALUE]]`, which is the *second* value in the list.

Since the linked list is in a loop, this doesn't actually cause problems, because the first value will eventually be reached. However, this behavior is counter-intuitive and could be prevented by initializing the traversal one step backward.

Recommendation: For the `replace()` and `remove()` functions, consider initializing the `cursor` to `SENTINEL_VALUE` instead of `self[SENTINEL_VALUE]`.

Clave: Fixed with commits 2bba7a5e and ed64e82e.

Cantina Managed: Verified.

3.5.8 Revert for wrong input

Severity: Informational

Context: ERC20Paymaster.sol#L60-L63

Description: `ERC20Paymaster`'s constructor skips for tokens for which wrong data has been passed (zero address or out-of-range `priceMarkup`):

```
constructor(TokenInput[] memory tokens) {
    for (uint256 i = 0; i < tokens.length; i++) {
        // Skip zero-addresses
        if (tokens[i].tokenAddress == address(0)) continue;

        // Skip false markup values
        if (tokens[i].priceMarkup < 5000 || tokens[i].priceMarkup >= 100000) continue;
        uint192 priceMarkup = uint192(tokens[i].priceMarkup * (MARKUP_NOMINATOR / 1e4));

        allowedTokens[tokens[i].tokenAddress] = TokenData(tokens[i].decimals, priceMarkup);
    }
}
```

It's better to revert for incorrect arguments as a safety measure. It may indicate towards a problem with how those arguments were generated.

Recommendation: Revert instead of skipping tokens with incorrect data:

```
  // Skip zero-addresses
- if (tokens[i].tokenAddress == address(0)) continue;
+ if (tokens[i].tokenAddress == address(0)) revert;

  // Skip false markup values
- if (tokens[i].priceMarkup < 5000 || tokens[i].priceMarkup >= 100000) continue;
+ if (tokens[i].priceMarkup < 5000 || tokens[i].priceMarkup >= 100000) revert;
```

Clave: Fixed in PR 716.

Cantina Managed: Verified.

3.5.9 Incorrect Natspec

Severity: Informational

Context: ERC20Paymaster.sol#L226-L230

Description: `callOracle()` returns `uint256` but its Natspec says it returns `uint256[]`.

Recommendation: Fix the Natspec.

Clave: Fixed in PR 716.

Cantina Managed: Verified.

3.5.10 Unused import

Severity: Informational

Context: EOAValidator.sol#L5

Description: Following import is not used and can be removed:

```
import {Transaction} from '@matterlabs/zksync-contracts/l2/system-contracts/libraries/TransactionHelper.sol';
```

Recommendation: Remove the import.

Clave: Fixed with commit 9b22c453.

Cantina Managed: Verified.