SHERLOCK

Security Review For Sodax

Collaborative Audit Prepared For: **Sodax**
Lead Security Expert(s): **ge6a**
KupiaSec
Date Audited: **July 14 - July 19, 2025**

Introduction

The audit covers all core EVM contracts used by the Sodax protocol, including: - Relay Connection Contracts – for managing cross-chain messaging and execution - Wallet Abstraction Contracts – for unified wallet interaction across chains - Asset Manager Contracts – for handling assets and execution flows - Rate Limiter Contracts – for enforcing transaction and withdrawal limits - Spoke Asset Management Contracts – for asset handling on connected chains

Scope

Repository: icon-project/intent-relay

Audited Commit: 5c45d2370c9bca6db4d4f82ce75063146454033b

Final Commit: a725b5be30631f9c4703b1b95e843069f1f6e365

Files:

- contracts/evm/contracts/connectionv3/ConnectionV3.sol
- contracts/evm/contracts/connectionv3/Utils.sol
- contracts/evm/library/interfaces/IConnectionV3.sol

Repository: icon-project/sodax-contracts

Audited Commit: f9e3116bc4ffc9c46d1042efea32dbea81462fd4

Final Commit: 3e37825b892bdd0e2257aa5dd3410d70a7bc37ec

Files:

- evm/contracts/assetManager/AssetManager.sol
- evm/contracts/assetManager/AssetToken.sol
- evm/contracts/assetManager/TransferLib.sol
- evm/contracts/spokeAssetManager/RateLimit.sol
- evm/contracts/spokeAssetManager/SpokeAssetManager.sol
- evm/contracts/spokesManager/RateLimitMessage.sol
- evm/contracts/utils/AddressLib.sol
- evm/contracts/utils/FeeM.sol
- evm/contracts/wallet/WalletFactory.sol
- evm/contracts/wallet/Wallet.sol

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
3	9	5

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: The Utils.getMessageHash() function is vulnerable to the hash collision attack.

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/14

Summary

The `Utils.getMessageHash()` function uses the `abi.encodePacked`. In addition it uses the `trimmedBeBytes()` function to removing leading zeros. So, hash collision is possible.

Vulnerability Detail

The `Utils.getMessageHash()` function uses the `abi.encodePacked`.

```
function getMessageHash(
    uint256 srcChainId,
    bytes memory srcAddress,
    uint256 _connSn,
    bytes memory _msg,
    uint256 dstChainId,
    bytes memory dappAddr

) internal pure returns (bytes32) {
    bytes memory encoded = abi
        .encodePacked(
            trimmedBeBytes(srcChainId),
            srcAddress,
            trimmedBeBytes(_connSn),
            _msg,
            trimmedBeBytes(dstChainId),
            dappAddr
        );
    return keccak256(encoded);
}
```

In addition it uses the `trimmedBeBytes()` function to removing leading zeros. So, hash collision is possible.

```
function trimmedBeBytes(uint256 number) public pure returns (bytes memory) {
    bytes memory temp = abi.encodePacked(number);
    uint256 startIndex = 0;
    while (startIndex < temp.length && temp[startIndex] == 0) {
        startIndex++;
    }
    bytes memory result = new bytes(temp.length - startIndex);
    for (uint256 i = 0; i < result.length; i++) {
```

```
        result[i] = temp[startIndex + i];
    }
    return result;
}
```

As a result, an attacker can make a completely different malicious message which has the same digest with an illegal message.

Impact

An attacker can use an illegal signature for his illegal message. An attacker may also render all nonces invalid, effectively preventing the protocol from functioning as intended.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/main/intent-relay/contracts/evm/contracts/connectionv3/Utils.sol#L37

Tool Used

Manual Review

Recommendation

`abi.encodePacked()` and the function `trimmedBeBytes()` should not be used in the `Utils.getMessageHash()` function.

Issue H-2: No withdraw fees function in ConnectionV3

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/23

Vulnerability Detail

In the `AssetManager` and `SpokeAssetManager` contracts, the `transfer` functions send native tokens via `ConnectionV3.sendMessage`, which are likely intended to cover relayer costs. However, there is no function allowing administrators to withdraw these funds, so they remain locked in the contract. The only way to retrieve them is through a contract upgrade.

Impact

Loss of funds from the fees.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/assetManager/AssetManager.sol#L200

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/spokeAssetManager/SpokeAssetManager.sol#L95

Tool Used

Manual Review

Recommendation

Implement withdraw function in the `ConnectionV3` contract

Issue H-3: Inconsistent `assetInfo` mapping after token override allows invalid transfers

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/25

Summary

When a token is overridden, the `assetInfo` mapping is not updated accordingly. The old entry in `assetInfo` remains accessible, can still be used for transfers and a new entry is not added. This could lead to inconsistencies, since the same token on the spoke is now associated with two different assets on the hub.

Vulnerability Detail

Let's take a look at the implementation of the `transfer` function.

```
function transfer(
        address token,
        bytes calldata to,
        uint256 amount,
        bytes calldata data
    ) external payable override {
        AssetInfo memory info = assetInfo[token];
        IAssetToken(token).burnFrom(msg.sender, amount);

        Transfer memory _transfer = Transfer(
            info.spokeAddress,
            msg.sender.toBytes(),
            to,
            amount,
            data
        );

        bytes memory manager = spokeAssetManager[info.chainID];
        connection.sendMessage{value: msg.value}(
            info.chainID,
            manager,
            _transfer.encode()
        );
    }
```

In it, based on the wrapped token we want to send, the info for `spokeAddress` and `chainId` is retrieved from the `assetInfo` map. However, when the token is changed using `overrideToken`, no entry is made for the new address in `assetInfo`. Therefore, when the `transfer` function is called, the `chainId` and `assetInfo` will be empty.

In the same scenario, if a user calls `transfer` with the address of the old token (the one before `overrideToken` was called), it will burn a certain amount of it and initiate a transfer to the spoke, which should not be possible, because that token has already been overridden by another one.

Impact

Broken functionality and potential double spending, which leads to loss of funds.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/assetManager/AssetManager.sol#L98-L104

Tool Used

Manual Review

Recommendation

In `overrideToken`, a new entry should be added to `assetInfo` for the new asset, and the old entry should be removed.

Issue M-1: `IConnectionV3.sendMessage()` is not payable.

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/16

Summary

Any calls to `ConnectionV3.sendMessage()` involve value transfer. The calls are made through the `IConnectionV3 interface`, so `IConnectionV3.sendMessage()` should be payable.

Vulnerability Detail

`IConnectionV3.sendMessage()` is not marked as `payable`. As a result, any calls to `ConnectionV3.sendMessage()` that are made through the `IConnectionV3` interface will revert.

```
        function sendMessage(
            uint256 dstChainId,
            bytes memory dstAddress,
            bytes memory payload
@>      ) external;
```

Impact

Transfers are not possible.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/main/intent-relay/contracts/evm/library/interfaces/IConnectionV3.sol#L17

Tool Used

Manual Review

Recommendation

```
        function sendMessage(
            uint256 dstChainId,
            bytes memory dstAddress,
            bytes memory payload
-       ) external;
+       ) external payable;
```

Issue M-2: `RateLimit.setRateLimit()` does not account for the accrued amount generated by the previous `ratePerSecond`

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/17

Summary

The `RateLimit.setRateLimit()` function doesn't update `config.available` by the accrued amount generated by the previous `ratePerSecond`.

Vulnerability Detail

In `RateLimit.setRateLimit()`, the function doesn't update `config.available` if it is non-zero. Since some time has elapsed since the last update, the accrued amount generated by the old `ratePerSecond` should be added to the available balance.

```
    function setRateLimit(
        address token,
        uint256 ratePerSecond,
        uint256 maxAllowedWithdrawal
    ) internal {
        require(ratePerSecond > 0, "InvalidRateLimit");

        RateLimitConfig storage config = tokenConfigs[token];
        config.ratePerSecond = ratePerSecond;
        config.maxAvailable = maxAllowedWithdrawal;
        config.lastUpdated = block.timestamp;
        config.lastRecordedBalance = balanceOf(token);
@>      if (config.available == 0) {
            config.available = maxAllowedWithdrawal;
        }

        emit TokenConfigUpdated(token, ratePerSecond, maxAllowedWithdrawal);
    }
```

Impact

The available amount is not correctly updated, leading to incorrect rate limiting behavior.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/main/sodax-contracts/evm/contracts/spokeAssetManager/RateLimit.sol#L115

Tool Used

Manual Review

Recommendation

Should account for the accrued amount generated by the previous `ratePerSecond`.

Issue M-3: `ConnectionV3.sendMessage()` is not marked as payable, making transfers impossible.

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/18

Summary

Any calls to `ConnectionV3.sendMessage()` involve value transfer, so it should be payable.

Vulnerability Detail

Since `ConnectionV3.sendMessage()` is not payable, it cannot accept any Ether.

```
ConnectionV3.sol

        function sendMessage(
            uint256 dstChainId,
            bytes memory dstAddress,
            bytes memory payload
@>      ) external override {
            ...
        }
```

However, both `AssetManager.transfer()` and `SpokeAssetManager.transfer()` attempt to call `ConnectionV3.sendMessage()` with a specified value. Consequently, these transfer attempts will revert.

```
AssetManager.sol

    function transfer(
        ...

@>      connection.sendMessage{value: msg.value}(
            info.chainID,
            manager,
            _transfer.encode()
        );
    }

-----------------------------------

SpokeAssetManager.sol

    function transfer(
        ...
```

```
@>      connection.sendMessage{value: messageFee}(
            hubChainId,
            hubAssetManager,
            _transfer.encode()
        );

    }
```

Impact

Transfers are not possible.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/main/intent-relay/contracts/evm/contracts/connectionv3/ConnectionV3.sol#L82

Tool Used

Manual Review

Recommendation

```
        function sendMessage(
            uint256 dstChainId,
            bytes memory dstAddress,
            bytes memory payload
-       ) external override {
+       ) external override payable {
            ...
        }
```

Issue M-4: Lack of consideration of fee on transfer token

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/21

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The protocol aims to support bridging any token, but it inadequately considers fee-on-transfer tokens, resulting in the protocol receiving fewer tokens than expected.

Vulnerability Detail

https://github.com/sherlock-audit/2025-07-sodax-july-14th/tree/main/sodax-contracts/README.md#35

```
* Any token should be able to be bridged to any other connected chain
```

Impact

The loss are accumulated, resulting users being unable to withdraw due to insufficient asset.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/tree/main/sodax-contracts/evm/contracts/spokeAssetManager/SpokeAssetManager.sol#L80

Tool Used

Manual Review

Recommendation

Issue M-5: Lack of consideration of rebasing token

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/22

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The protocol aims to support the bridging of any token, but it inadequately considers rebasing tokens, which results in the locking of funds in connectors on spoke chains. As rebasing tokens are escrowed in the connectors, their balance increases over time. However, the corresponding balance on the hub chain does not increase. Therefore, when users on the hub chain attempt to withdraw the tokens, they cannot fully withdraw the amount they previously escrowed. It causes lock of funds.

Vulnerability Detail

https://github.com/sherlock-audit/2025-07-sodax-july-14th/tree/main/sodax-contracts/README.md#35

```
* Any token should be able to be bridged to any other connected chain
```

Impact

Rebasing tokens are locked in the connectors on spoke chains.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/tree/main/sodax-contracts/evm/contracts/spokeAssetManager/SpokeAssetManager.sol#L80

Tool Used

Manual Review

Recommendation

Issue M-6: Relayers are vulnerable to message flooding

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/24

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

The message sending functions, such as `AssetManager.transfer`, lack access control, meaning anyone can create messages without restriction, potentially flooding relayers

with messages and delaying the processing of legitimate ones. These functions support optional fees (e.g., in `AssetManager.transfer`), but the fees are not enforced.

In `ConnectionV3.sendMessage`, an event is emitted for each message, but the paid fee is not recorded in the event. As a result, it's impossible to filter messages based on the fee amount using only the event logs.

Impact

Increased relayer costs and potential delays / denial of service for legitimate users.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/assetManager/AssetManager.sol#L182-L187

Tool Used

Manual Review

Recommendation

A possible mitigation is to implement a mechanism that makes such flooding attacks economically unfeasible, for example by requiring mandatory onchain fees.

Discussion

AntonAndell

For now we think normal gas fees is sufficient to stop spam on most chains. Our relayer does not do event listening but get filtered on demand of users directly. But that still makes spamming viable but we do not deem it a large risk atm

Issue M-7: Replay attack on `RateLimit` configuration via reusable signed payloads

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/26

Summary

The `SpokesManager.sendMessage` function lacks access control, and authentication relies solely on signatures included in the message payload. Since payloads become publicly known after first use, an attacker can replay them. This allows unauthorized reuse of old configurations or repeated resets of the rate limit, undermining the mechanism's intended protections.

Vulnerability Detail

Changes to the configuration of the `RateLimit` contract are made by sending a message from the hub via the `SpokesManager` contract. The `sendMessage` function is used, and it has no access control. Authentication of the account sending the new configuration is done via a signature of the data structure being sent. In `RateLimit.recvMessage`, the address of the signer is extracted and checked against a list of stored addresses in the `RateLimit` contract (e.g., hub admin, signers, etc.). The rest of the message verification process follows the standard implementation in `ConnectionV3`.

The issue is that once a message with a given `payload` (i.e., signature and data) is sent, it becomes publicly known. A malicious user can reuse this same signature and data and call `SpokesManager.sendMessage` to send a new message with the same payload. This message will pass verification and be executed successfully. As a result, old payloads can be reused to repeatedly change the configuration of `RateLimit` in ways not intended by the administrator. This can also be used to reset the rate limit, undermining the entire purpose of the mechanism.

Currently the only defense is the deadline but it is set to 1 hours which is enough time to reset the rate limit multiple times.

Impact

An attacker can repeatedly replay valid, signed configuration messages to reset or alter the `RateLimit` state.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/spokeAssetManager/RateLimit.sol#L172-L188

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/spokesManager/SpokesManager.sol#L39-L44

Tool Used

Manual Review

Recommendation

Add a mechanism to invalidate the signed payload after it has been used.

Issue M-8: RateLimit configuration message can affect tokens on unintended chains

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/27

Vulnerability Detail

According to the protocol design, a configuration update message for the RateLimit contract sent from the Hub is expected to be executable on every Spoke. This works well for changes like updating the admin or signers, but causes problems when resetting or modifying the rate limit for a specific token, because the same address can represent a valid token on different chains.

For example, some tokens like USDC intentionally deploy wrapped versions of their tokens at the same address across multiple chains to simplify user interactions. As a result, if a rate limit reset or update is performed on one chain, it could also affect the token with the same address on another chain – even if that was not the administrator's intention.

Impact

Broken core functionality and ability for malicious user to reset rate limit on a chain which was not intended to be affected.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/spokeAssetManager/RateLimit.sol#L198-L215

Tool Used

Manual Review

Recommendation

A possible solution is to make these 2 types of messages unique for each (chain, token) pair.

Issue M-9: Incorrect reset of `config.available` on fully used rate limit

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/28

Vulnerability Detail

When the `RateLimit.setRateLimit` function is called and `config.available == 0`, `config.available` is set to `maxAllowedWithdrawal`. The intention behind this is to initialize `config.available` to the maximum value in case the rate limit hasn't been set before. However, it's possible that a limit has already been configured but is simply fully used at the time of the call, which is why `config.available` is 0. In that case, resetting `config.available` to the maximum value `maxAllowedWithdrawal` is incorrect. If a reset is necessary, the explicit `resetRateLimit` function can be used instead.

Impact

Wrong `config.avaialbe` in some cases

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/spokeAssetManager/RateLimit.sol#L115-L117

Tool Used

Manual Review

Recommendation

Adjust the logic to set `config.available` to maxAllowedWithdrawal only if it is really a new rate limit and not an update.

Issue L-1: `ConnectionV3.setValidatorThreshold()` does not verify whether the new threshold is less than the current number of validators.

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/19

Summary

`validatorsThreshold` should not exceed the total number of validators, so resetting `validatorsThreshold` should verify this condition.

Vulnerability Detail

`validatorsThreshold` should not exceed the total number of validators. However, `setValidatorThreshold()` doesn't check if the new threshold meets this requirement.

```
function setValidatorThreshold(uint8 _count) external onlyOwner {
    require(_count > 0, "Invalid threshold");
    validatorsThreshold = _count;
}
```

Impact

`validatorsThreshold` might exceed the total number of validators, making signature verification impossible.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/main/intent-relay/contracts/evm/contracts/connectionv3/ConnectionV3.sol#L166

Tool Used

Manual Review

Recommendation

Implement a check to ensure the new `validatorsThreshold` doesn't exceed the total number of validators.

Issue L-2: Data location mismatch in `IConnectionV3.verifyMessage()`

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/20

Vulnerability Detail

In `IConnectionV3.verifyMessage()`, the parameter `payload` is defined with the data location `calldata`. However, in `ConnectionV3.verifyMessage()`, the same parameter is defined with the data location `memory`.

```
interface IConnectionV3 {
    ...

    function verifyMessage(
        uint256 srcChainId,
        bytes calldata srcAddress,
        uint256 connSn,
@>      bytes calldata payload,
        bytes[] calldata signatures
    ) external;
}

-----------------------------------

contract ConnectionV3 is IConnectionV3, UUPSUpgradeable, OwnableUpgradeable {
    ...

    function verifyMessage(
        uint256 srcChainId,
        bytes calldata srcAddress,
        uint256 _connSn,
@>      bytes memory _payload,
        bytes[] calldata signatures
    ) public override {
        ...
}
```

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/main/intent-relay/contracts/evm/library/interfaces/IConnectionV3.sol#L31

Tool Used

Manual Review

Recommendation

```
    function verifyMessage(
        uint256 srcChainId,
        bytes calldata srcAddress,
        uint256 connSn,
-       bytes calldata payload,
+       bytes memory payload,
        bytes[] calldata signatures
    ) external;
```

Issue L-3: Revert of `ConnectionV3.updateValidators` with more than 300 validators

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/29

Vulnerability Detail

On each iteration of the for loop in the `ConnectionV3.updateValidators` function, all previously added validators are traversed, which is not optimal. This way, with around 300 validators, the gas usage will exceed 30M. If the protocol is expected to operate with more `validators` in some time of its lifespan, it won't be possible to add them.

Impact

Possible DoS of core functionality

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/intent-relay/contracts/evm/contracts/connectionv3/ConnectionV3.sol#L53-L60

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/intent-relay/contracts/evm/contracts/connectionv3/ConnectionV3.sol#L71-L75

Tool Used

Manual Review

Recommendation

Optimize the nested loops using a map.

Issue L-4: Unauthorized execution of stored wallet actions

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/30

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

The `Wallet` contract includes functionality that allows a message to be stored for later execution. This is done by saving the hash of the message's payload in `storedCalls`. When a user wants to execute the stored message, they can call the `executeStored` function with the corresponding data whose hash matches an entry in `storedCalls`.

The `executeStored` function has no access control and can be called by anyone, as long as they know the data of a previously stored call.

The problem arises when a user wants to perform the same action more than once. After the first execution, the call data becomes publicly known, which allows anyone to re-execute the action without authorization, even if the user did not intend for it to be executed again at that moment.

Additionally, it is important to note that a malicious user may also attempt to guess the data behind a given hash, which - in certain situations with enough context – is not difficult.

If we assume the stored call should be executed asap, then there's no issue. But it's possible that users don't want to execute it immediately. For example, a user may want to perform some action from the wallet at the right moment but without the delay from the relayer. They could store the call data in advance and execute it instantly when the time is right.

Impact

Execution of an action on behalf of a user at specific moment without their consent.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/wallet/Wallet.sol#L99-L103

Tool Used

Manual Review

Recommendation

Add a unique variable in each data that produce a hash.

Issue L-5: Re-entrancy in `Wallet.executeStored`

Source: https://github.com/sherlock-audit/2025-07-sodax-july-14th/issues/31

Vulnerability Detail

The Wallet.executeStored function is used to execute calls whose hash has been previously stored in storedCalls. The problem is that the this.executeCalls function, which processes the calls, is executed before the corresponding entry is deleted from storedCalls. This allows a re-entrancy attack to be performed, enabling the stored calls to be executed multiple times. A limitation of this attack is that the user must have initiated the storage of the hash for such calls, making it a user mistake.

Impact

A user can be tricked into making a call to a potentially malicious contract, which can then perform a re-entrancy attack.

Code Snippet

https://github.com/sherlock-audit/2025-07-sodax-july-14th/blob/e50953220e8bbe737598cb108c91500faefa7f04/sodax-contracts/evm/contracts/wallet/Wallet.sol#L99-L103

Tool Used

Manual Review

Recommendation

Delete the record from `storedCalls` before the execution of `this.executeCalls`

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.