SHERLOCK

Security Review For Sodax

Collaborative Audit Prepared For: **Sodax**
Lead Security Expert(s): **0xeix**
KupiaSec
Date Audited: **October 27 - November 3, 2025**

Introduction

The relay architecture forms a key component of the SodaX infrastructure, enabling secure and scalable coordination of cross-chain transaction validation, signing, and execution. It leverages AWS services such as SNS, SQS, Lambda, DynamoDB, and RDS to manage event-driven workflows and maintain system state. The design ensures segregation of responsibilities across verification, signing, and execution layers, with strict IAM-based controls governing access to sensitive components like signers. This setup provides both operational efficiency and strong security assurances, supporting reliable cross-chain interoperability within the SodaX system.

Scope

Repository: icon-project/intent-relay

Audited Commit: 4025178e8468f2e39fe456863798caf76f790d9b

Final Commit: 3613a8ae63e9d6047f44fe88d167eca98cd14b68

Files:

- apps/dynamodb_lambda/db/db_service.go
- apps/dynamodb_lambda/main.go
- apps/ext_updater/main.go
- apps/kms_sign/main.go
- apps/kms_sign/server.go
- apps/kms_sign/signer/common.go
- apps/kms_sign/utils/dynamo_svc.go
- apps/kms_sign/utils/types.go
- verifiers/common/src/axios-utils.ts
- verifiers/common/src/b64-utils.ts
- verifiers/common/src/hex-utils.ts
- verifiers/common/src/lambda-utils.ts
- verifiers/cosmos/infrastructure/zip.ts
- verifiers/cosmos/src/config.ts
- verifiers/cosmos/src/index.ts
- verifiers/cosmos/src/processor.ts
- verifiers/cosmos/src/server.ts
- verifiers/cosmos/src/types.ts

- verifiers/cosmos/src/utils.ts
- verifiers/evm/infrastructure/zip.ts
- verifiers/evm/src/config.ts
- verifiers/evm/src/index.ts
- verifiers/evm/src/server.ts
- verifiers/evm/src/types.ts
- verifiers/icon/infrastructure/zip.ts
- verifiers/icon/src/config.ts
- verifiers/icon/src/index.ts
- verifiers/icon/src/server.ts
- verifiers/icon/src/types.ts
- verifiers/solana/infrastructure/zip.ts
- verifiers/solana/src/config.ts
- verifiers/solana/src/index.ts
- verifiers/solana/src/server.ts
- verifiers/solana/src/types.ts
- verifiers/solana/src/utils.ts
- verifiers/stellar/infrastructure/zip.ts
- verifiers/stellar/src/config.ts
- verifiers/stellar/src/index.ts
- verifiers/stellar/src/server.ts
- verifiers/stellar/src/types.ts
- verifiers/stellar/src/utils.ts
- verifiers/sui/infrastructure/zip.ts
- verifiers/sui/src/config.ts
- verifiers/sui/src/index.ts
- verifiers/sui/src/server.ts
- verifiers/sui/src/types.ts

Final Commit Hash

3613a8ae63e9d6047f44fe88d167eca98cd14b68

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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.
- 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
2	4	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Solana Off-Chain Log Processing Vulnerability: `connContractFound` State Not Reset [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/47

Summary

In the Solana off-chain log processing code, the `connContractFound` flag is used to determine whether subsequent logs belong to the `connection` program before decoding SendMessage events. However, the flag is never reset after a transaction completes. This can allow logs from other programs in the same transaction to be incorrectly processed as if they were from the `connection` program.

Vulnerability Detail

The off-chain function `parseAndProcessLogs` processes logs emitted by a transaction (txHash) and looks for `SendMessage` events emitted by the connection program:

```
export async function parseAndProcessLogs(txHash, data) {
    ... ...
}
```

Within the function, a boolean flag `connContractFound` is used: `connContractFound` is set to true when a log emitted by the connection program is detected.

```
let connContractFound = false
for (let log of parsedResponse.result.meta.logMessages) {
    if (log.startsWith(`Program ${config.CONNECTION_ADDRESS}`)) {
    connContractFound = true
```

However, the flag is never reset, either per instruction or after encountering a success or failed log.

```
if (log.startsWith(eventLogPrefix) && connContractFound) {
```

As a result, any subsequent logs in the same transaction–even if they originate from unrelated programs–can pass the `connContractFound` check.

An attacker could:

1. Submit a transaction that contains:

- A legitimate send_message instruction from the connection program.
- Malicious instructions from other programs that emit fake logs matching eventLogPrefix.

2. The off-chain processor may interpret these fake logs as legitimate SendMessage events because connContractFound remains true.

Impact

- Off-chain consumers may process fake messages or arbitrary payloads.
- Could lead to unauthorized message signing, fund manipulation, or other downstream effects depending
- on how the processed messages are used. Violates the integrity of cross-chain or inter-program message handling.

Code Snippet

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/verifiers/solana/src/index.ts#L46-L122

Tool Used

Manual Review

Recommendation

1. Reset connContractFound appropriately:
- Per instruction: Check whether the log belongs to the connection program for each instruction separately.
- On transaction completion: Reset the flag after a success or failed log.
2. Alternatively, match logs to program IDs per instruction rather than maintaining a global boolean across all logs in a transaction.

Discussion

pragyanshrestha-ibriz

Fix PR: https://github.com/icon-project/intent-relay/pull/293

Issue H-2: Risk of Seed Collision in Solana Connection Program [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/55

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

Summary

The Solana connection program derives its receipt account PDA using trimmed big-endian bytes of `src_chain_id` and `conn_sn` as seeds. This introduces a risk of seed collisions, as different numeric combinations may produce identical concatenated seed sequences after trimming leading zeros.

Vulnerability Detail

The `receipt` account is initialized with seeds derived as follows:

```
#[account(
    init,
    payer = signer,
    seeds = [Receipt::SEED_PREFIX.as_bytes(), &trimmed_be_bytes(src_chain_id),
    ↪  &trimmed_be_bytes(conn_sn)],
    space = Receipt::LEN,
    bump
)]
pub receipt: Account<'info, Receipt>
```

Here, the `trimmed_be_bytes()` function removes leading zeros from the byte representation of integers. As a result, two distinct `(src_chain_id, conn_sn)` pairs can lead to the same concatenated seed, for example:

Consider the following case: trimmed_be_bytes(src_chain_id) = [0x01, 0x02], trimmed_be_bytes(conn_sn)=[0x03] trimmed_be_bytes(src_chain_id) = [0x01], trimmed_be_bytes(conn_sn)=[0x02, 0x03] In both cases, the seed arrays are identical, causing a PDA collision.

When this happens, the second transaction attempting to initialize a receipt account will revert permanently, since `#[account(init)]` enforces that the derived PDA must not already exist on-chain.

Impact

Permanent denial of service for specific (src_chain_id, conn_sn) pairs due to seed collision. Message verification or bridging failures for affected transactions.

Code Snippet

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/contracts/solana/programs/connection/src/contexts.rs#L85-L92

Tool Used

Manual Review

Recommendation

Use fixed-length encoded bytes (e.g., 8 or 16 bytes) for seed derivation instead of trimmed representations to ensure uniqueness and avoid collisions.

Discussion

pragyanshrestha-ibriz

Seems not an issue for our use case. Seeing only the code it seems both src_chain_id and sn are variable, in that case issue is valid. but in our case src_chain_id is not a variable, so that its not a issue. Any change in this will result in another serious issue also

KupiaSecAdmin

Why isn't `src_chain_id` variable? Different source chains have different ids.

pragyanshrestha-ibriz

> Why isn't `src_chain_id` variable? Different source chains have different ids.

for a spoke chain the src_chain_id is always the hub_chain_Id in our case and that is not changeable the validation for it is in the contracts that calls the connection contract

Issue M-1: `isTxFinal()` is not called if the number of blocks to wait is equal to zero skipping the default case [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/48

Summary

The function `isTxFinal()` is skipped while there's a default value to be provided for such a scenario.

Vulnerability Detail

Consider the `isTxFinal()` function implementation:

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/verifiers/evm/src/index.ts#L38-47

```
async function isTxFinal(receipt: ethers.TransactionReceipt, minConfirmations = 12)
↪  {
  try {
    const latestBlockPromises = providers.map(provider =>
    ↪  provider.getBlockNumber());
    const latestBlock = await Promise.any(latestBlockPromises);
    const confirmations = latestBlock - receipt.blockNumber;
    return confirmations >= minConfirmations;
  }
  catch { }
  return false
}
```

As you can see here, the default value is supposed to be `12`:

```
minConfirmations = 12
```

However, when we actually call it inside of the `parseAndProcessLogs()`:

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/verifiers/evm/src/index.ts#L64-69

```
if (blocksToWait > 0) {
```

```
  const isFinal = await isTxFinal(receipt, blocksToWait)

  if (!isFinal) {
    throw new Error('Tx not finalized yet.');
  }
}
```

It skips the finalization checks instead of providing nothing and invoking the default case with `minConfirmations`. `blocksToWait` is always either value from `env` or `0`:

```
const blocksToWait = Number.parseInt(process.env.BLOCKS_TO_WAIT) || 0
```

Impact

The finalization check is skipped even though it's supposed to be triggered for the default value when the `blocksToWait` param is either undefined or equal to zero.

Tool Used

Manual Review

Recommendation

Change the logic so that the default scenario with `minConfirmations` is handled instead of being skipped.

Discussion

pragyanshrestha-ibriz

This will be handled by config updates once we have these wait block values from ticket - https://github.com/icon-project/intent-relay/issues/299.

bcsainju

We have created internal tickets to analyze appropriate block wait times for each EVM chains and then the param will be updated via config(the env variable BLOCKS_T0_WAIT). We have chosen this instead of simply going for a default value of greater than 0

Issue M-2: Returning status 200 even if there's an error updating the DB [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/50

Summary

Currently, in the non-docker path, we mistakenly log the DynamoDB error and returning `http.StatusOK` when in reality it wasn't updated.

Vulnerability Detail

Let's take a look at the non-docker path implementation:

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/apps/kms_sign/signer/common.go#L241-247

```
    er := dbService.UpsertDbEntry(requestBody.SrcChainId.String(),
    ↪  requestBody.DstChainId.String(),

        txnHashSn, signatures, string(bytesReq), hex.EncodeToString(msgHash),
        ↪  tableName, signingKeyParam)



    if er != nil {
        log.Println("error updating dynamodb", er.Error())
    }


    return GetReturnMessage("message signed", signature, http.StatusOK, true), nil
}
```

It can be seen that the error is just logged and instead of returning the message. For example, an error in the `UpdateDbEntry()` can be the following (when calling `UpdateItem()`):

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/apps/ext_updater/main.go#L76

```
_, err := dbc.UpdateItem(context.Background(), &dynamodb.UpdateItemInput
```

```
result, metadata, err := c.invokeOperation(ctx, "UpdateItem", params, optFns,
↪  c.addOperationUpdateItemMiddlewares)
    if err != nil {
        return nil, err
    }
```

So when we're invoking operation, different types of errors can be returned such as `SpanStatusError` or `OperationError`.

Impact

The caller believes the signature and metadata are persisted while the database never recorded them.

Tool Used

Manual Review

Recommendation

Consider returning

```
return GetReturnMessage("response error occurred: "+strconv.Itoa(resp.StatusCode),
↪  signature, http.StatusInternalServerError, false), nil
```

similar to how it's done in the docker path.

Issue M-3: Duplicate `axios.get()` Call in `subscribeUrl()` Causes Incorrect Error Handling [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/51

Summary

The `subscribeUrl()` function in `lambda-utils.ts` redundantly calls `axios.get(subscriptionUrl)` both inside and outside of the try block. This leads to a duplicate HTTP request and ineffective error handling, as the second request is made within the catch block without proper containment.

Vulnerability Detail

In the `subscribeUrl()` implementation, the same `axios.get(subscriptionUrl)` call is executed twice – once in the try block and again inside the catch block.

```
export const subscribeUrl= async (subscriptionUrl: string) => {
    try {
        await axios.get(subscriptionUrl);
        console.log('Subscription confirmed',subscriptionUrl);
    } catch (err) {
        await axios.get(subscriptionUrl);
        console.error('Error confirming subscription:',subscriptionUrl, err);
    }
}
```

This results in two issues:

Duplicate network request: The GET request is executed again in the catch block, which is unnecessary and may cause duplicate subscription attempts or race conditions if the endpoint has side effects.

Improper error handling: Because the second `axios.get()` is executed within the catch but outside any nested error handling, any exception from the second call will escape the try/catch block and result in an unhandled promise rejection or runtime failure.

Impact

Can result in uncaught exceptions if the second request fails.

Code Snippet

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/verifiers/common/src/lambda-utils.ts#L30-L38

Tool Used

Manual Review

Recommendation

```
        export const subscribeUrl= async (subscriptionUrl: string) => {
            try {
                await axios.get(subscriptionUrl);
                console.log('Subscription confirmed',subscriptionUrl);
            } catch (err) {
-               await axios.get(subscriptionUrl);
                console.error('Error confirming subscription:',subscriptionUrl,
                ↪  err);
            }
        }
```

Issue M-4: Errors when publishing the messages to SNS are ignored [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/54

Summary

When publishing the message to SNS, an error can occur that's ignored inside of the `handleAppendRequest()`.

Vulnerability Detail

After appending an attribute, there's an attempt to publish a message to SNS:

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/apps/dynamodb_lambda/main.go#L80-87

```
if request.RecvTxnHash != "" {
    if err := dbService.AppendAttribute(tableName, request.SourceChainId,
    ↪  request.TxnHashSn,
        "recvTxnHash", request.RecvTxnHash); err != nil {
        return getReturnMessage("failed adding attribute to db",
        ↪  http.StatusBadRequest, false), nil
    }


    publishToSNS(request.DstChainId, request.RecvTxnHash)
}
```

The problem is that the `publishToSNS()` function can return an error that's currently ignored:

```
if err != nil {
    fmt.Println(fmt.Printf("Failed to publish message: %v", err.Error()))
    return
}
```

Impact

Silent errors when publishing to SNS that are not accounted for in the system. Publishing SNS can be a part of the contract (e.g., other services advance state only when they see this event).

Tool Used

Manual Review

Recommendation

Either fail the request (and let your retry/queue do its job) or record the failure for replay.

Issue L-1: Improper error handling when creating a new request [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/49

Summary

A possible error in the `http.NewRequest` is not handled in the `HandleSignRequest()`.

Vulnerability Detail

At the moment, it's possible for the `NewRequest()` function to return an error, for example, when parsing:

```
u, err := urlpkg.Parse(url)
    if err != nil {
        return nil, err
    }
```

This will result eventually result in a nil-pointer panic.

Impact

A panic bypasses your normal error mapping and you don't have a clean 500 error with a meaningful JSON body as a return value plus potential cold starts.

Code Snippet

Tool Used

Manual Review

Recommendation

```
req, err := http.NewRequest("POST", apiURL, bytes.NewBuffer(jsonData))
if err != nil {
    return GetReturnMessage("error occurred: "+err.Error(), signature,
    ↪  http.StatusInternalServerError, false), nil
}
```

Issue L-2: No timeout specified for the client [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/52

Summary

The client is created with no timeout being specified.

Vulnerability Detail

By default, `http.Client{}` has `Timeout: 0` and the code builds the default one:

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/apps/kms_sign/signer/common.go#L229-230

```
client := &http.Client{}
resp, err := client.Do(req)
```

And `Do()` creates a deadline by using this client's `Timeout` value:

```
deadline = c.deadline()
```

```
func (c *Client) deadline() time.Time {
    if c.Timeout > 0 {
        return time.Now().Add(c.Timeout)
    }
    return time.Time{}
}
```

Impact

A request can hang indefinitely at any stage.

Tool Used

Manual Review

Recommendation

Introduce the following logic:

```
client := &http.Client{ Timeout: 10 * time.Second }
```

or something similar with the usage of `context`:

```
ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
```

Issue L-3: Attribute is viewed as added even if both `ExecTxnHash` and `RecvTxnHash` are equal to empty strings [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/issues/53

Summary

Both hashes are checked to be non-empty and if not, we append the attribute.

Vulnerability Detail

Take a look at the current functionality:

https://github.com/sherlock-audit/2025-10-sodax-intent-relay-oct-27th/blob/main/intent-relay/apps/dynamodb_lambda/main.go#L80-96

```
func handleAppendRequest(ctx context.Context, request DbRequest)
↪  (events.LambdaFunctionURLResponse, error) {



    if request.RecvTxnHash != "" {

        // append attribute
        if err := dbService.AppendAttribute(tableName, request.SourceChainId,
        ↪  request.TxnHashSn,
            "recvTxnHash", request.RecvTxnHash); err != nil {
            return getReturnMessage("failed adding attribute to db",
            ↪  http.StatusBadRequest, false), nil
        }


        publishToSNS(request.DstChainId, request.RecvTxnHash)
    }


    if request.ExecTxnHash != "" {



        if err := dbService.AppendAttribute(tableName, request.SourceChainId,
        ↪  request.TxnHashSn,

            "execTxnHash", request.ExecTxnHash); err != nil {
```

```
            return getReturnMessage("failed adding attribute to db",
            ↪  http.StatusBadRequest, false), nil
        }


        publishToSNS(request.DstChainId, request.ExecTxnHash)

    }

    return getReturnMessage("attribute added", http.StatusOK, true), nil


}
```

As you can see here, if one of the hashes is not empty, we `AppendAttribute()` the following way. However, the problem is that even if both are empty strings, we still mistakenly return:

```
return getReturnMessage("attribute added", http.StatusOK, true), nil
```

Impact

Clients think the state advanced when in reality it's not.

Tool Used

Manual Review

Recommendation

Consider adding:

```
return getReturnMessage("failed adding attribute to db", http.StatusBadRequest,
↪  false), nil
```

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.