CD SECURITY

AUDIT REPORT

Dexlyn Bridge
December 2024

Prepared by
0xluk3
MoveJay

Introduction

A time-boxed security review of the **Dexlyn** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Dexlyn**

Dexlyn Bridge is a blockchain interoperability protocol that enables secure cross-chain messaging and token transfers between different blockchain networks. The Aptos implementation of Dexlyn uses Move smart contracts to handle message routing, verification, and delivery. The protocol employs a modular security model where messages are verified by Interchain Security Modules (ISMs) and relies on a network of off-chain relayers to physically transmit messages between chains. Security is enforced through a decentralized network of validators who sign cross-chain messages ensuring their validity.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 663e74dea779bd48c29ab28f57372b73927833f1**

Scope

The following folders were in scope of the audit:

- `examples`
- `igps`
- `isms`
- `library`
- `mailbox`
- `router`
- `synthetic-tokens`
- `tokens`
- `validator-announce`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 2 issues
- Low & Info: 9 issues

Findings Summary

ID	Title	Severity	Status
[M-01]	Disabled router enrollment check allows processing of unroutable messages	Medium	Fixed
[M-02]	Destination token decimals cannot be updated after initial setting	Medium	Fixed
[L-01]	Friend feature is unused	Low	Fixed
[L-02]	Off by one error may cause some transfers to revert	Low	Acknowledged
[L-03]	Redundant decimal checks with inconsistent error handling	Low	Fixed
[L-04]	Insufficient validation in validator set configuration	Low	Fixed
[L-05]	Possible zero-amount transfers when downscaling decimals	Low	Fixed
[L-06]	Redundant Message Tracking State Variables	Low	Acknowledged
[I-01]	Misleading initialization function name	Informational	Fixed
[I-02]	Empty module	Informational	Fixed
[I-03]	One-way ownership transfer	Informational	Acknowledged

Detailed Findings

[M-01] Disabled router enrollment check allows processing of unroutable messages

Severity

Impact: High

Likelihood: Low

Description

In hyperlane-monorepo/move/mailbox/sources/mailbox.move L136, the handle_message function has a commented out router enrollment check:

```
// router::assert_router_should_be_enrolled<T>(src_domain, sender_addr);
```

This check would have verified that a router exists for the message's source domain. Without it, messages from domains without enrolled routers can be processed through inbox_process(), but will likely become stuck as they cannot be routed to their final destination. This wastes resources and could lead to permanently stranded messages.

Recommendations

Uncomment and restore the router enrollment verification

[M-02] Destination token decimals cannot be updated after initial setting

Severity

Impact: High

Likelihood: Low

Description

In `hyperlane-monorepo/move/synthetic-tokens/sources/hyper_coin.move` and `hyper_coin_collateral.move`, in the `set_destination_token_decimal` function, the use of `table::add` is designed to abort if an entry for the key already exists, as per the Move Table implementation

Once a decimal value is set for a destination domain, any attempt to modify it will cause the transaction to revert. This provides no mechanism to update decimals in case of misconfiguration or legitimate changes. On the other hand, this could be needed for example in case of a mistake, or an upgrade of the target token on the remote destination.

Recommendations

Use `table::upsert` instead of `table::add` if decimal modifications should be supported.

[L-01] Friend feature is unused

Description

In the following files:

```
./router/sources/events.move
./mailbox/sources/events.move
./validator-announce/sources/events.move
./igps/sources/events.move
```

Friend modules as per the docs can be used to control access to specific functions within a module that defines the friend. However, in the above cases, all callable functions are public anyway, which either symbolized friend feature being unused, or they are missing additional logic.

On the other hand, those functions return only certain structs, which do not pose a risk to the protocol even if called by an unauthorized actor.

Recommendations

In order to use the friend feature, the public(friend) visibility instead of public, for the functions to be callable only by friend modules.

[L-02] Off by one error may cause some transfers to revert

Description

In hyperlane-monorepo/move/igps/sources/igps.move L61, there is the following assertion to ensure a user is able to pay for gas:

```
    // check account's balance if it is enough to pay interchain gas
    assert!(coin::balance<AptosCoin>(account_address) > required_amount,
ERROR_INSUFFICIENT_INTERCHAIN_GAS);
```

However, if a user has only the exact required amount, this will not work since the module requires greater than the amount. This may cause user transactions to fail even if they shouldn't, limiting user'sw ability to use the protocol and diminishing user experience.

Recommendations

Use >= greater or equal instead.

[L-03] Redundant decimal checks with inconsistent error handling

Description

In `hyper_coin.move` and `hyper_coin_collateral.move`, the `transfer_remote` function contains two identical checks for destination decimals:

```
assert!(table::contains(&state.destination_decimals, dest_domain), 2);  // Uses ESOURCE_DECIMAL_NOT_SET
assert!(table::contains(&state.destination_decimals,dest_domain), error::not_found(EDESTINATION_DECIMAL_NOT_SET));
```

The first check uses error code 2 (defined as `ESOURCE_DECIMAL_NOT_SET`), which is misleading since it's checking destination decimals. The second check uses the correct error constant but is redundant. This redundancy increases gas costs and the error inconsistency could confuse developers debugging issues.

It could also mean that some check was missed, that should have been there instead. However, the source decimals are part of the coin itself, which does not need to be checked.

Recommendations

Remove the first check and keep only the semantically correct one with `EDESTINATION_DECIMAL_NOT_SET`. If source decimal validation is needed, add it separately with the appropriate error code.

[L-04] Insufficient validation in validator set configuration

Description

In `hyperlane-monorepo/move/isms/sources/multisig_ism.move`, the `set_validators_and_threshold` function lacks important validation checks when configuring validators and thresholds:

- the function permits duplicate validator addresses
- it allows arbitrary threshold values as long as they are greater than 0 and don't exceed the validator count

This could lead to misconfigured security settings where the same validator is counted multiple times or where the threshold is set too low for adequate security

Recommendations

Implement duplicate address detection in the validator vector. Consider enforcing a minimum threshold of `(validator_count / 2) + 1` to ensure proper security.

[L-05] Possible zero-amount transfers when downscaling decimals

Description

In `hyper_coin_collateral.move` and `hyper_coin.move`, when transferring tokens between chains with different decimal places, small amounts can be rounded down to zero when the source chain has higher precision than the destination:

```
else {
    data_amount = (amount as u256) / calculate_power(10, ((source_decimals
- destination_decimals) as u16));
    amount = (data_amount as u64); // Can become 0 for small amounts
};
```

Since Aptos does not prevent zero-amount transfers by default, this can lead to:

- gas being spent to process a message that will ultimately fail,
- potential inconsistencies or undefined behavior on the receiving chain when attempting to mint zero tokens,
- user funds being burned on the source chain without equivalent minting on the destination

Recommendations

Add a validation check after decimal conversion to ensure the final amount is greater than zero:

```
assert!(data_amount > 0, EAMOUNT_TOO_SMALL);
```

[L-06] Redundant Message Tracking State Variables

Description

The contract maintains redundant state variables for message tracking that are either unused or duplicate functionality already provided by the mailbox module. Specifically:

- **received_messages**: `vector<vector<u8>>`
    - Never used in the contract except for initialization.
    - Mailbox already provides message tracking.
    - Takes up unnecessary storage space.

- **last_id**: `vector<u8>`
    - Only used in a view function.
    - Mailbox already handles message verification and tracking.
    - Adds unnecessary state management.

The mailbox module already provides comprehensive message handling, including:

- Message replay protection via `delivered` map.
- Message verification through validator signatures.
- Domain and version checks.
- Merkle tree tracking of messages.

Impact

- Unnecessary gas costs for state storage.
- Redundant code complexity.
- No functional impact as mailbox handles message tracking.

[I-01] Misleading initialization function name

Description

In hyperlane-monorepo/move/mailbox/sources/mailbox.move L77, the initialize function can be called multiple times by the owner to change the domain:

```
public entry fun initialize(
    account: &signer,
    domain: u32,
) acquires MailBoxState {
    assert_owner_address(signer::address_of(account));
    let state = borrow_global_mut<MailBoxState>(@hp_mailbox);
    state.local_domain = domain;
}
```

The function name initialize implies a one-time setup operation, but this function actually acts as a setter that can modify the local_domain value repeatedly. This creates confusion, as initialization functions are conventionally expected to be called only once during contract setup.

This naming inconsistency makes the code less maintainable and could lead to misunderstandings about the function's intended use.

Recommendations

Rename the function to better reflect its actual behavior for example `set_domain`, or make it one-time if intended.

[I-02] Empty module

Description

Module `hyperlane-monorepo/move/isms/sources/events.move` is empty, and contains only the module declaration. This unnecessarily increases the complexity of the code.

Recommendations

Remove the module if it is not used.

[I-03] One-way ownership transfer

Description

In `router/sources/router.move` and `mailbox/sources/mailbox.move`, the ownership transfer pattern is unidirectional, allowing the current owner to directly transfer ownership without acceptance from the new owner. This creates risk as ownership could be transferred to an invalid or unprepared address, potentially resulting in a permanent loss of control.

Recommendations

Implement a two-step ownership transfer pattern where the current owner initiates the transfer by proposing a new owner, and the proposed owner must explicitly accept ownership through a separate transaction.