RESPONSIBLE DISCLOSURE

Packet Forward Middleware (PFM)

Prepared by SCV-Security on
18th July 2023

Introduction

During an internal security review of a specific contract functionality involving packet-forward-middleware (PFM) and IBC-hooks, SCV team has identified a notable aspect that has caught our attention. It's important to note that the SCV did not conduct a comprehensive audit of these modules but focused on examining a specific vulnerability case scenario.

Before diving into more details, it would be relevant to first define what each implementation does and what problems they individually solve.

The packet-forward-middleware (PFM) is a middleware IBC module for Cosmos that routes IBC packets from a source chain to a destination chain. A chain which incorporates the PFM module is able to route incoming IBC packets to a destination chain (multi-hop routing). The middle-chains that forward the packages to the final destination can also be called counterparty chains.

The IBC-hooks, also known as wasm-hook, is an IBC middleware which is used to allow ICS-20 token transfers to initiate contract calls. This allows cross-chain contract calls that involve token movement.

Now we know both modules definitions, let's dive in!

About the Vulnerability

To facilitate the understanding, let's assume the following case scenario:

Let's suppose a single wasm smart-contract contract acting similar to an interchain account having the ability to execute messages using ibc-hooks from chainA to chainB (A <> B).

When messages from chainA arrives on chainB, the IBC-Hooks module creates a new variation account representing the sender of the transaction, in this case, from chainA to chainB based on the following logic:

- Bech32(Hash("ibc-wasm-hook-intermediary" || channelID || sender))

Where:

- **ibc-wasm-hook-intermediary**: module prefix string
- **channelID**: is the channel-id on the local chain

- **sender**: address that initiated the transaction on chainA

The different address variation is needed because the sender of an IBC packet in a counterparty chain (middle-chain) is not trusted and has the full ability to lie about it. This is also documented here.

The IBC-hooks approach to mitigate potential counterparty attacks relies on the channel-id on the local chain, which provides a level of trustworthiness. This is because malicious middle-chains can impersonate addresses, but they cannot impersonate the local channel on the receiver chain since it is pre-established and controlled by the receiver chain.

IBC-hooks can lead to more implementation cases where developers can use IBC messages from chainA to perform actions on chainB.

An example could be an Automatic Market Maker (AMM) contract that accepts token deposits via IBC from a specific chain. In this example, a smart-contract performs logical operations based on the address variation established through the hook.

Furthermore, a different scenario could involve a contract deployed on chainB, but its control is governed by chainA. This is achieved through the utilisation of IBC-hooks, which allows chainA to influence and manage the operations of the contract residing on chainB.

Since there is no distinction between an IBC and a PFM packet, it becomes possible to manipulate the memo field in a way that triggers both PFM and IBC-hooks on the destination chain over the same “trustworthy” channel.

The vulnerability arises from the fact that PFM middleware, responsible for transferring (forward-packets) data between different chains, **allows any address to be designated as the `receiver` in the middle-chain arriving at the destination as the sender**.

This means that the address variation implemented by the IBC-hooks module, which is intended to prevent impersonations, becomes ineffective and untrustworthy at the destination. As a result, an attacker can exploit this vulnerability without requiring access to the sender’s key by using a counterparty chain with PFM targeting a destination chain.

Technical Details

The ICS-20 specification omits user authentication data but fully covers authentication information for the entire chain where it permits the transfer of tokens from one place to another. Essentially, an ICS-20 packet serves as an authorization from the source chain to the final destination.

As IBC does not inherently ensure a trustworthy topology, middle-chains have the potential to tamper with data. This fact is well-known, and applications must consider it when incorporating logic like callbacks upon an IBC packet at the receiving side. Handling such scenarios carefully is crucial to maintain data integrity and security.

In reality, these implementations are typically applied at the state-machine level (core) rather than the application layer (smart contracts). One user-case example is Stride, where it permits staking tokens via IBC. As a precautionary measure, IBC staking has been temporarily disabled as a precautionary measure due to this finding. Refer to proposal #209 for further details.

IBC-hooks, in contrast, are implemented at the application level, providing developers with the flexibility to apply their own logic when handling received IBC packets. The use of wasm-hooks enables various user cases, including IBC cross-chain token swaps and interchain applications. This empowers developers to create innovative solutions that leverage the potential of IBC for seamless cross-chain interactions via wasm smart contracts.

In smart contract development, relying on fundamental functionality like identifying the sender to determine logical operations is more than a common case. However, as already described, this approach can lead to significant problems and potential vulnerabilities due to PFM context.

For example, if a `MsgTransfer` packet carrying a memo field is set to trigger the ibc-hooks is sent from Terra chain from the address `terra123` to Juno chain to the address `juno456` using the `channel-0` on Terra, that has it counterparty on Juno as `channel-1`, the `info.sender` contract state will be computed as:
`Bech32(Hash("ibc-wasm-hook-intermediary" || channel-1/terra123))`

However, the memo can be set to trigger PFM on an intermediary chain, without any distinction between a conventional IBC message. Using Osmosis as an intermediary chain for example, it is possible to set the memo field to trigger both PFM and ibc-hooks on the destination chain as per case below:

- From the Terra chain, using the `terra123` address .
- forward-packet to Osmosis chain, to the `osmo456` address using `channel-0` on Terra that has it counterparty on Osmosis as `channel-1`
- to Juno to contract `juno789` using `channel-2` on Osmosis that has it counterparty on Juno as `channel-3`

When the ibc-hook is triggered on Juno, the `info.sender` will be compute as: `Bech32(Hash("ibc-wasm-hook-intermediary" || channel-3/osmo456))`

This example demonstrates that the attacker does not need to hold `osmo456` keys by using the PFM from a bounce on Osmosis originally initiated on Terra.

Note that, an attacker can use different routes in order to exploit contracts functionality and asserts.

Practical Attack

Assuming the following context below:

- **Terra contract address**: `terra123`
- **Terra contract functionality:** only allows calls from osmo123 (via the hook computed with `bech32(Hash("ibc-wasm-hook-intermediary", "osmo123", "channel-1"))` that results to an "interchain" address `terra999`.
- **Terra contract Interface:** `ExecuteMsg::PingMe {} => ping_me(deps, env, info)`
- **Juno attacker wallet**: `juno1337`
- **Exploit method**: IBC-Hooks + PFM on the routes, Juno -> Osmosis -> Terra.

Juno attacker `juno1337` can successfully call `PingMe()` on `terra123` by crafting an IBC message using Osmosis as counterparty as per following:

```
receiver: osmo123
sender: juno1337
channel: (not relevant for this vulnerability)
memo: (this will trigger PFM)
{"forward":{
   "receiver": "terra123",
   "channel-id": "channel-1"
   ....
   "next":{
      "wasm":{
         "contract": "terra123",
         "msg": MsgExecute::PingMe.to_json()
         }
      }
}
```

When the packet arrive on Osmosis, the PFM module redirect this packet to terra, and the packet will be:

```
receiver: terra123
sender: osmosis123
channel: channel-1
```

Which makes it possible to confuse the `info.sender` as PFM will forward the package as follow:

- **sender**: receive of the packet arrive into the middle chain
- **receiver**: address specified on memo field of previous packet
- **channel-id**: channel specified on memo field of previous packet

Recommendations

It is advised not to rely on the raw sender of an IBC packet without further verification checks and assurances. This includes any contract operations or contract asserts that utilise the IBC-hook contract-state (`info.sender`) imposed by the IBC-hooks.

Potential Remediations

Remediation measures were recommended and thoroughly discussed. However, due to potential compatibility issues, their implementation is currently uncertain. Please note that the following remediations are provided for reference and context, not in any weight-in priority order or as an endorsement

- PFM could create a new sender account on middle-chains using a hash and bech32 encoding, ensuring integrity at the final destination, regardless of the number of hops:
    - `Bech32(hash(port+channel+original_sender))`

- IBC-Hooks could implement an AllowList vector (with no PFM supported chains) where chains individually select trustworthy channels they want to trigger actions from.

- IBC to imply a topology of trust chain routes.

- Sign user transaction at origin with the desired action and pass along the route.

- Consider the use and adoption of ICS-999.

Conclusion

In conclusion, middle-chains are considerably not trustworthy by default since the protocol does not imply any specification on the authenticity of packets permitting data tampering. While remediation measures were recommended, implementation remains uncertain due to compatibility issues. It is advised not to solely rely on the raw sender of an IBC packet including the address variation imposed by IBC-hooks, and careful handling of IBC scenarios is essential for data integrity and security.

Special Thanks

On behalf the SCV team, we would like to thank individuals and teams that assisted us in this disclosure and directly collaborated:

- **Rhaki**, an SCV contributor and the original reporter who found this vulnerability.
- **BigLabs** team, for providing a reliable testing environment and contract samples.
- **Notional Venture**, for helping us fetching teams directly and verified the vulnerability.
- **Osmosis** team, for promptly verified and acknowledged this vulnerability and helping by fetching other teams directly.
- **Stride** team, for the prompt verification on their side and quick action protecting users funds on proposal #209.