Interchain Foundation IBC-Go Channel Upgrade Feature Assessment

Security Assessment Report

Prepared for Interchain Foundation
March 7, 2024 (version 1.0)

Project Team:

Technical Testing	Brandon Perry and James Cook
Technical Editing	Ryan Hanson and Molly Vukusich
Project Management	Sara Bettes

Table of Contents

Engagement Overview

Assessment Components and Objectives

Interchain Foundation ("Interchain Foundation") recently engaged Atredis Partners ("Atredis") to perform an IBC-Go Channel Upgrade Feature Assessment of the Interchain Foundation platform. Objectives included validation that Interchain Foundation infrastructure and services were developed and deployed with security best practices in mind, and to obtain third party validation that any significant vulnerabilities present in Interchain Foundation's environment were identified for remediation.

Testing was performed from January 29 through February 16, 2024, by Brandon Perry and James Cook of the Atredis Partners team, with Sara Bettes providing project management and delivery oversight. For Atredis Partners' assessment methodology, please see Appendix I of this document, and for team biographies, please see Appendix II. Specific testing components and testing tasks are included below.

COMPONENT	ENGAGEMENT TASKS
Interchain Foundation IBC-Go Channel Upgrade Feature Assessment	

	• Source-assisted penetration testing of the channel upgrade feature in ibc-go
	  ○ Review of bootstrapping and feature source code
	  ○ Manual runtime and protocol analysis
	  ○ Development of adversarial test cases where feasible
	  ○ Fuzzing and fault injection
	  ○ PoC generation and validation of findings
	  ○ Targeted exploitation of identified high-risk vulnerabilities
	  ○ PoC generation and validation of findings

Reporting and Analysis	
Analysis and Deliverables	• Status Reporting and Realtime Communication • Comprehensive Engagement Deliverable • Engagement Outbrief and Remediation Review

The ultimate goal of the assessment was to provide a clear picture of risks, vulnerabilities, and exposures as they relate to accepted security best practices, such as those created by the National Institute of Standards and Technology (NIST), Open Web Application Security Project (OWASP), or the Center for Internet Security (CIS). Augmenting these, Atredis Partners also draws on its extensive experience in secure development and in testing high-criticality applications and advanced exploitation.

Engagement Tasks

Atredis Partners performed the following tasks, at a high level, for in-scope targets during the engagement.

Application Penetration Testing

For relevant web applications, APIs, and web services, Atredis performed automated and manual application penetration testing of these components, applying generally accepted testing best practices as derived from OWASP and the Web Application Security Consortium (WASC).

Testing was performed from the perspective of an anonymous intruder, identifying scenarios from the perspective of an opportunistic, Internet-based threat actor with no knowledge of the environment, as well as, from the perspective a user working to laterally move through the environment to bypass security restrictions and user access levels.

Where relevant, Atredis Partners utilized both automated fuzzing and fault injection frameworks as well as purpose-built, task-specific testing tools tailored to the application and platforms under review.

Binary and Runtime Analysis

For relevant software targets identified during the course of this engagement, Atredis performed binary and runtime analysis, using debugging and decompilation tools to analyze application flow to aid in software security analysis. Where relevant, purpose-built tools such as fuzzers and customized network clients were utilized to aid in vulnerability identification.

Network Protocol Analysis

With the objective of identifying scenarios where the integrity of trusted communications can be diminished or reduced, Atredis Partners reviewed network traffic using various packet flow analysis and packet capture tools to observe in-scope network traffic. Network communications were analyzed for the presence of cleartext communications or scenarios where the integrity of cryptographic communications can be diminished, and Atredis attempted to identify means to bypass or circumvent network authentication or replay communications, as well as other case-dependent means to abuse the environment to disrupt, intercept, or otherwise negatively affect in-scope targets and communications.

Source Code Analysis

Atredis reviewed the in-scope application source code, with an eye for security-relevant software defects. To aid in vulnerability discovery, application components were mapped out and modeled until a thorough understanding of execution flow, code paths, and application design and architecture is obtained. To aid in this process, the assessment team engaged key stakeholders and members of the development team where possible to provide structured walkthroughs and interviews, helping the team rapidly gain an understanding of the application's design and development lifecycle.

Executive Summary

Atredis was tasked with testing the IBC Go channel upgrade feature using the Hermes[1] relayer. IBC Go developers provided Atredis with the main git repository for the IBC Go project on Github. Using this repository, Atredis was able to build and run the channel upgrade features and tests defined as the scope of the engagement.

The testing team performed the channel upgrade process on local chains running on Atredis hardware. Atredis proxied all communications between the relayer and the chains, which allowed great power for testing replay, out-of-order delivery, or other attacks. The channel upgrade feature strictly enforces certain channel states during the upgrade which prevented several types of attacks.

Atredis focused on several facets of the upgrade process. First, Atredis was able to focus on initiating a channel upgrade without a proper vote passing on the chains. Atredis focused on how potentially malicious version strings in proposals could affect upgrades if passed with a vote. Atredis also looked at how a potentially malicious relayer or compromised relayer could affect the chains and the upgrade process.

Key Conclusions

Overall, Atredis Partners found Interchain Foundation's channel upgrade feature well-architected from a security perspective. Atredis verified several key security layers in the channel upgrade implementation through manual code review and testing. Atredis found no way to begin a channel upgrade without both chains passing a vote to initiate the channel upgrade.

Atredis Partners performed extensive manual runtime testing of Interchain Foundation's Channel Upgrade feature from both an application and network security perspective. Atredis approached attack scenarios from the position of several key actors, such as compromised relays, malicious proposals, out-of-order and replay attacks, and other similar styles of attacks. Atredis found no way to leave the channel in a partially upgraded state or to have the chains vote and approve different parameters for the same channel upgrade. Three informational issues were ultimately noted during testing.

As in any security assessment, some general areas for improvement were noted, but overall, Atredis Partners would rate Interchain Foundation's platform as strong from a security perspective and generally well-aligned with modern asymmetric encryption and signing when integrating with blockchain technology.

[1] https://github.com/informalsystems/hermes

Platform Overview

Channels allow two independent chains to communicate with an agreed upon set of parameters. In order to allow channels to be upgraded as new or different parameters are needed, IBC Go has implemented a channel upgrade flow for channels which ensures authentication and authorization across chains. The channel upgrade functions for the IBC Go project are implemented across several files within the source code.

```
bperry@bperry-ThinkPad-X1-Carbon-6th:~/tmp/ibc-go/e2e$ find .. | grep upgrade.go
../modules/core/04-channel/keeper/upgrade.go
../modules/core/04-channel/types/upgrade.go
../modules/light-clients/08-wasm/types/upgrade.go
../modules/light-clients/07-tendermint/upgrade.go
```

Upgrade Implementation Files

Two independent chains may use a relayer like Hermes to create interchain channels for communication. The channel is voted on and created by a proposal, a set of parameters, and version settings. To initiate a channel upgrade, both chains must pass a vote on a proposal to upgrade the channel. The process of performing a channel upgrade can be implemented with any relayer in any language (and is expected to be added to other relayer implementations), Hermes is the first relayer to support the IBC channel upgrade process directly. To perform a channel upgrade, two chains with an active channel must vote on an upgrade proposal. The 'for' votes must pass the chain's defined threshold within the voting period.

```
{
  "title": "Channel upgrade init",
  "summary": "Channel upgrade init",
  "messages": [
  {
         "@type": "/ibc.core.channel.v1.MsgChannelUpgradeInit",
         "signer": "cosmos10d07y265gmmuvt4z0w9aw880jnsr700j6zn9kn",
         "port_id": "transfer",
         "channel_id": "channel-0",
         "fields": {
             "ordering": "ORDER_UNORDERED",
             "connection_hops": ["connection-0"],
             "version": "{\"fee_version\":\"ics29-1\",\"app_version\":\"ics20-1\"}"
          }
   }
   ],
   "metadata": "AQ==",
   "deposit": "100005stake"
}
```

Example Proposal JSON Stanza with Channel Upgrade Message

To begin testing, Atredis focused on the well-written unit and end-to-end tests maintained by the IBC-Go developers.

```
TestV8ToV8_1ChainUpgrade
TestV7ToV8ChainUpgrade
TestV7ToV7_1ChainUpgrade
TestV6ToV7ChainUpgrade
TestSendEnabledParam
TestScheduleIBCUpgrade_Succeeds
TestRecoverClient_Succeeds_GrandpaContract
TestRecoverClient_Succeeds
TestReceiveEnabledParam
TestPayPacketFeeAsync_SingleSender_NoCounterPartyAddress
TestMultiMsg_MsgPayPacketFeeSingleSender
TestMsgTransfer_WithMemo
TestMsgTransfer_TimesOut_GrandpaContract
TestMsgTransfer_Timeout_Nonincentivized
TestMsgTransfer_Succeeds_Nonincentivized
TestMsgTransfer_Succeeds_GrandpaContract
TestMsgTransfer_Localhost
TestMsgTransfer_Fails_InvalidAddress
TestMsgSendTx_SuccessfulTransfer_UnorderedChannel
TestMsgSendTx_SuccessfulTransfer_AfterReopeningICA
TestMsgSendTx_SuccessfulTransfer
TestMsgSendTx_SuccessfulBankSend_Incentivized
TestMsgSendTx_FailedTransfer_InsufficientFunds
TestMsgSendTx_FailedBankSend_Incentivized
TestMsgPayPacketFee_SingleSender_TimesOut
TestMsgPayPacketFee_InvalidReceiverAccount
TestMsgPayPacketFee_AsyncSingleSender_Succeeds
TestMsgPayPacketFee_AsyncMultipleSenders_Succeeds
TestMsgMigrateContract_Success_GrandpaContract
TestMsgMigrateContract_ContractError_GrandpaContract
TestMaxExpectedTimePerBlockParam
TestInterchainAccounts_ReopenChannel_Localhost
TestInterchainAccounts_Localhost
TestInterchainAccountsGroupsIntegration
TestInterchainAccountsGovIntegration
TestIBCWasmChainUpgrade
TestIBCGenesis
TestIBCChainUpgrade
TestHostEnabledParam
TestControllerEnabledParam
TestClient_Update_Misbehaviour
TestClientUpdateProposal_Succeeds
> TestChannelUpgrade_WithFeeMiddleware_Succeeds
TestChannelUpgrade_WithFeeMiddleware_FailsWithTimeoutOnAck
TestChainUpgrade
TestAuthz_MsgTransfer_Succeeds
TestAuthz_InvalidTransferAuthorizations
TestAllowedClientsParam
48/48
```

End-to-End Tests for Channel Upgrade Feature

This enabled Atredis to quickly begin understanding the upgrade flows and what issues had been previously found and fixed. While some testing was done by altering tests directly, this was a cumbersome and time-intensive process.

```
32 // TestChannelUpgrade_WithFeeMiddleware_Succeeds tests upgrading a transfer channel to wire up fee middleware
33 func (s *ChannelTestSuite) TestChannelUpgrade_WithFeeMiddleware_Succeeds() {
34   t := s.T()
35   ctx := context.TODO()
36
37   relayer, channelA := s.SetupChainsRelayerAndChannel(ctx, s.TransferChannelOptions())
38   channelB := channelA.Counterparty
39   chainA, chainB := s.GetChains()
40
41   chainADenom := chainA.Config().Denom
42   chainBDenom := chainB.Config().Denom
43   chainAIBCToken := testsuite.GetIBCToken(chainBDenom, channelA.PortID, channelA.ChannelID)
44   chainBIBCToken := testsuite.GetIBCToken(chainADenom, channelB.PortID, channelB.ChannelID)
45
46   chainAWallet := s.CreateUserOnChainA(ctx, testvalues.StartingTokenAmount)
47   chainAAddress := chainAWallet.FormattedAddress()
48
49   chainBWallet := s.CreateUserOnChainB(ctx, testvalues.StartingTokenAmount)
50   chainBAddress := chainBWallet.FormattedAddress()
51
52   var (
53     chainARelayerWallet, chainBRelayerWallet ibc.Wallet
54     relayerAStartingBalance                  int64
55     testFee                                  = testvalues.DefaultFee(chainADenom)
56   )
57
58   s.Require().NoError(test.WaitForBlocks(ctx, 1, chainA, chainB), "failed to wait for blocks")
59
60   // trying to create some inflight packets, although they might get relayed before the upgrade starts
61   t.Run("create inflight transfer packets between chain A and chain B", func(t *testing.T) {
62     chainBWalletAmount := ibc.WalletAmount{
63       Address: chainBWallet.FormattedAddress(), // destination address
64       Denom:   chainADenom,
65       Amount:  sdkmath.NewInt(testvalues.IBCTransferAmount),
66     }
```

Example Test in upgrade_test.go

For instance, Atredis would remove key lines from the tests, or add them twice in a row, in order to replicate replay, out-of-order, or timeouts attacks at the test level.

```
  83   t.Run("execute gov proposal to initiate channel upgrade", func(t *testing.T) {
  84     chA, err := s.QueryChannel(ctx, chainA, channelA.PortID, channelA.ChannelID)
         85     s.Require().NoError(err)
         86
s(chA)) ~ 87     //s.initiateChannelUpgrade(ctx, chainA, chainAWallet, channelA.PortID, channelA.ChannelID, s.createUpgradeField
         88   })
         89
         90   t.Run("start relayer", func(t *testing.T) {
         91     s.StartRelayer(relayer)
         92   })
```

Commenting Out Key Lines in Test

```
bperry@bperry-ThinkPad-X1-Carbon-6th:~/tmp/ibc-go/e2e$ make e2e-test
./scripts/init.sh
using config file at /home/bperry/.ibc-go-e2e-config.yaml for e2e test
for id in  ; do \
    /usr/bin/docker stop $id ; \
    /usr/bin/docker rm $id ; \
done
./scripts/run-e2e.sh
# github.com/cosmos/ibc-go/e2e/tests/core/04-channel [github.com/cosmos/ibc-
go/e2e/tests/core/04-channel.test]
tests/core/04-channel/upgrades_test.go:84:3: chA declared and not used
FAILgithub.com/cosmos/ibc-go/e2e/tests/core/04-channel [build failed]
FAIL
make: *** [Makefile:14: e2e-test] Error 1
```

Test Failing After Modified Line

Atredis ultimately decided to spin up local chains for testing. Atredis made specific changes to the chains settings to facilitate testing. For instance, voting threshold was set to 30% vote is a pass, the voting period at 180 seconds, and the amount required to vote was reduced. These settings allowed Atredis to more quickly iterate during tests. The same configurations settings and sequences were done on the second chain as well. Atredis was able to use the `simd` utility to propose the channel upgrade and vote on both chains.

```
$ simd tx gov submit-proposal ./proposal.json --from $VALIDATOR_CHAIN1 --chain-id chain1 --
keyring-backend test --home ./testing/gm/chain1 --node http://localhost:27000

$ simd tx gov vote 1 yes --from $VALIDATOR_CHAIN1 --chain-id chain1 --keyring-backend test
--home ./testing/gm/chain1 --node http://localhost:27000
```

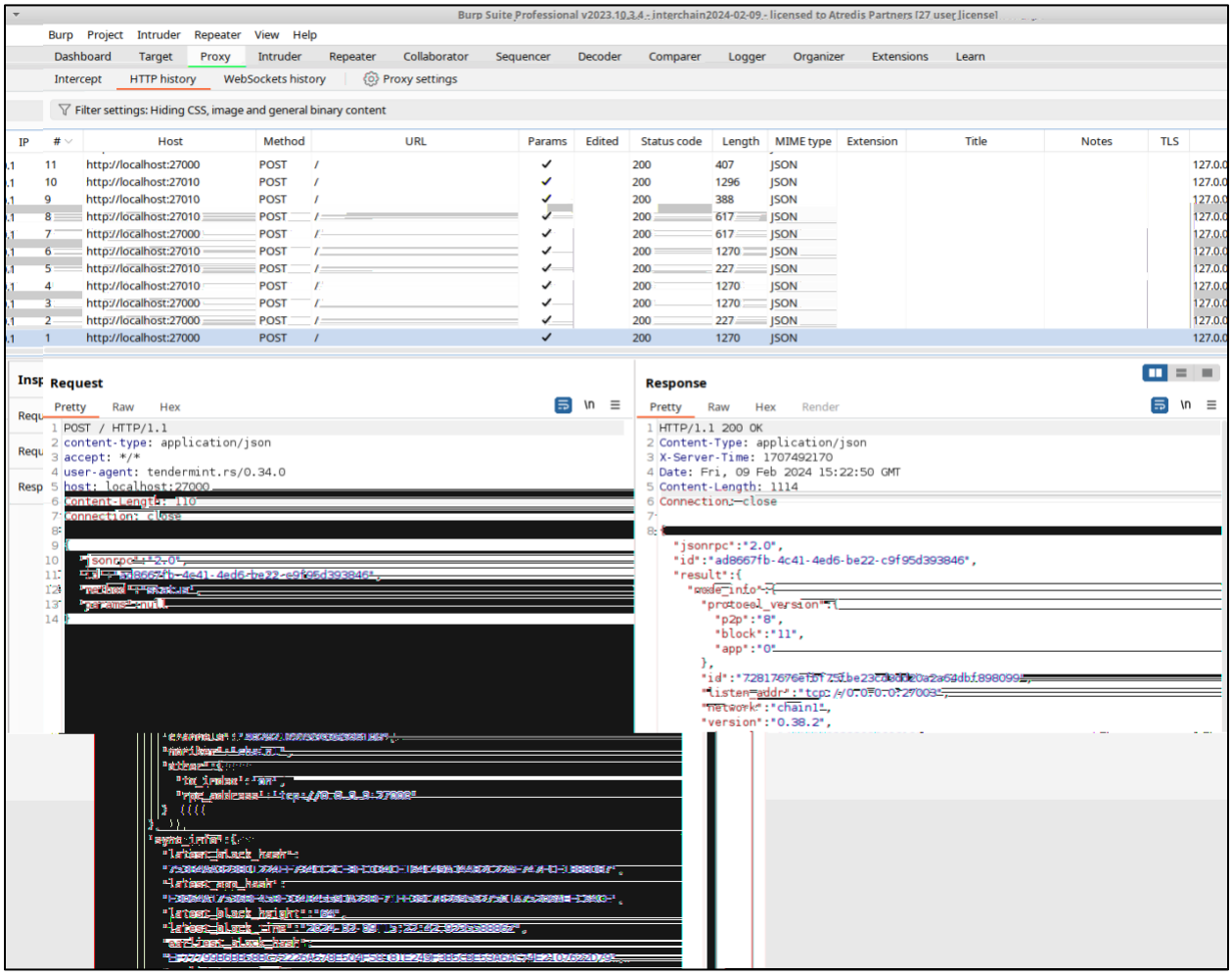
Upgrade Voting Process on Chain1

Hermes

Hermes is a stand-alone process (as opposed to a smart contract) written in Rust called a relayer. This external process watches chains and relays messages between chain RPCs (Remote Procedure Call) using pre-defined protocol parameters. In order to fully test the upgrade process, Atredis was able to proxy the RPC communications between the chain RPCs and the relayer process.

```
http_proxy=http://127.0.0.1:8080 hermes --config $HERMES_CONFIG_PATH start
```

Starting Hermes Relayer with HTTP Proxy



Proxying Hermes

Once started, Hermes begins making requests to the chain RPCs in order to gather the current state of the chains themselves.

```
POST / HTTP/1.1
content-type: application/json
accept: */*
user-agent: tendermint.rs/0.34.0
host: localhost:27000
Content-Length: 110
Connection: close

{
  "jsonrpc": "2.0",
  "id": "ad8667fb-4c41-4ed6-be22-c9f95d393846",
  "method": "status",
  "params": null
}
```

Example RPC Status Call Request

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Server-Time: 1707492170
Date: Fri, 09 Feb 2024 15:22:50 GMT
Content-Length: 1114
Connection: close


{"jsonrpc":"2.0","id":"ad8667fb-4c41-4ed6-be22-
c9f95d393846","result":{"node_info":{"protocol_version":{"p2p":"8","block":"11","app":"0"},
"id":"72817676efbf75fbe23cd3dd20a2a64dbf898099","listen_addr":"tcp://0.0.0.0:27003","networ
k":"chain1","version":"0.38.2","channels":"40202122233038606100","moniker":"chain1","other"
:{"tx_index":"on","rpc_address":"tcp://0.0.0.0:27000"}},"sync_info":{"latest_block_hash":"7
53B4AA3238D1224EF734CC2CE8FCD34CE164C49AD4482C22AE747FCE1888D87","latest_app_hash":"E3064A1
7536BF45BE334B4559DA23BF71FE39C7B2695627501A75206AEEC64CE","latest_block_height":"64","late
st_block_time":"2024-02-
09T15:22:42.923558886Z","earliest_block_hash":"EF77799B6BB58BC72226A578E504F58181E249F3B5CB
E53A6AC74E2107622D79","earliest_app_hash":"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934C
A495991B7852B855","earliest_block_height":"1","earliest_block_time":"2024-02-
09T15:15:17.602993026Z","catching_up":false},"validator_info":{"address":"03F43CE3F929A62CC
C649907DCEF1A214DB6D902","pub_key":{"type":"tendermint/PubKeyEd25519","value":"MVWx7YEvQd6w
l8vapgdoNwJEkRGLTpx0PNfbjM87w98="},"voting_power":"1"}}}
```

Example RPC Status Response with Metadata from Chain RPC

Once the requests were being captured, Atredis was able to find and decode the `protobuf` transaction packets which encode the actual channel messages. This allowed Atredis to perform replay or other attacks.

```
POST / HTTP/1.1
content-type: application/json
accept: */*
user-agent: tendermint.rs/0.34.0
host: localhost:27000
Content-Length: 1775
Connection: close



{
  "jsonrpc": "2.0",
  "id": "ad6b2b9c-30ec-4d4d-8349-44104af107e8",
  "method": "broadcast_tx_sync",
  "params": {
    "tx":
"CqEICuUHCiMvaWJjLmNvcmUuY2xpZW50LnYxLk1zZ1VwZGF0ZUNsaWVudBK9BwoPMDctdGVuZGVybWludC0wEvoGCi
YvaWJjLmxpZ2h0Y2xpZW50cy50ZW5kZXJtaW50LnYxLkhlYWRlchLPBgrIBAqMAwoCCAsSBmNoYWluMhhYIgwIvIeZr
gYQ3M7hswIqSAog7AT6XW76ccmiXv79ATwcblPWg9+jK8wl/ORXxKUlzdkSJAgBEiBdBDR2kRAIoSj74jcLxXF/olTR
ZMct4CekC53NE/nBbTIgGA4bFr3z/ibmE1UpcbrbtOtXgbfTeKJkp7Aa6L8DMzY6ICjQ4jZOc0zOa0iLa9fZQL+ldMQ
9rNT1cPpYHwoR9teBQiAEaC4hmRnUs7VGJkEZrwfNa8VZSdBchC13G6OBVc+S7UogBGguIZkZ1LO1RiZBGa8HzWvFWU
nQXIQtdxujgVXPku1SIASAkbx93Cg/d7+/kdc8RNpYw9+KnLyGdAXYt/ParaIvWiDBQbQhg6I9eend2bxLrAS7T/CsB
45gz9JhRA7P3HC0OmIg47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFVqIOOwxEKY/BwUmvv0yJlvuSQnrkHk
ZJuTTKSVmRt4UrhVchSPog3D+aWmGArI/l4km9vJcRd7whK2AQhYGkgKIII6XAhjqdV8wjucSrT2dF8dRAgeFJfWNbl
QI1MSikduEiQIARIgZpS2gkWPSavVjFTWXvfNU17Ei9gSPQpic7ht6HAMkp4iaAgCEhSPog3D+aWmGArI/l4km9vJcR
d7whoMCMGHma4GEOSG7sACIkAqkIYVNm9GQkMLlJ5XUTipuLFF7avWjDmerGU2dh/Plf9cHzb6Wc4W7FLTISu6fGSXb
pGBt5Un+OZ72a6PhWYHEn4KPAoUj6INw/mlphgKyP5eJJvbyXEXe8ISIgogAX0bC5lAVQaBv6HoRJxaRdR6Bd7QbPRO
uYGmognE/xMYARI8ChSPog3D+aWmGArI/l4km9vJcRd7whIiCiABfRsLmUBVBoG/oehEnFpF1HoF3tBs9E65gaaiCcT
/ExgBGAEaAhBXIn4KPAoUj6INw/mlphgKyP5eJJvbyXEXe8ISIgogAX0bC5lAVQaBv6HoRJxaRdR6Bd7QbPROuYGmog
nE/xMYARI8ChSPog3D+aWmGArI/l4km9vJcRd7whIiCiABfRsLmUBVBoG/oehEnFpF1HoF3tBs9E65gaaiCcT/ExgBG
AEaLWNvc21vczEyaHBtenZ1Zmt3Znh1NWp5azk1MmE0YTk4dWVubHozZ2UzanF0YRI3aGVybWVzIDEuOC4wKzkxZmY4
ZmJhIChodHRwczovL2hlcm1lcy5pbmZvcm1hbC5zeXN0ZW1zKRJmClAKRgofL2Nvc21vcy5jcnlwdG8uc2VjcDI1Nms
xLlB1YktleRIjCiECFY9es0STPxeb35klZdCYTaASlCMvzaaIM26KkO5OLA0SBAoCCAEYAhISCgwKBXN0YWtlEgMxMD
kQ684GGkBfTk7vKfa9w6SOd3IieT3QIcg5m/5VVpEDcMaccLnliiZcucQ1tTuhwKJbzRvXeVFbQor2zPFzi46/C1NMg
K+v"
  }
}
```

Example RPC Request with Protobuf Transaction Packet

The `protobuf` packets were able to be decoded, altered, and re-encoded by a command-line tool called `protoscope`[2].

[2] https://github.com/protocolbuffers/protoscope

```
bperry@bperry-ThinkPad-X1-Carbon-6th:~$ echo
'CqEICuUHCiMvaWJjLmNvcmUuY2xpZW50LnYxLk1zZ1VwZGF0ZUNsaWVudBK9BwoPMDctdGVuZGVybWludC0wEvoGCi
YvaWJjLmxpZ2h0Y2xpZW50cy50ZW5kZXJtaW50LnYxLkhlYWRlchLPBgrIBAqMAwoCCAsSBmNoYWluMhhYIgwIvIeZr
gYQ3M7hswIqSAog7AT6XW76ccmiXv79ATwcblPWg9+jK8wl/ORXxKUlzdkSJAgBEiBdBDR2kRAIoSj74jcLxXF/olTR
ZMct4CekC53NE/nBbTIgGA4bFr3z/ibmE1UpcbrbtOtXgbfTeKJkp7Aa6L8DMzY6ICjQ4jZOc0zOa0iLa9fZQL+ldMQ
9rNT1cPpYHwoR9teBQiAEaC4hmRnUs7VGJkEZrwfNa8VZSdBchC13G6OBVc+S7UogBGguIZkZ1LO1RiZBGa8HzWvFWU
nQXIQtdxujgVXPku1SIASAkbx93Cg/d7+/kdc8RNpYw9+KnLyGdAXYt/ParaIvWiDBQbQhg6I9eend2bxLrAS7T/CsB
45gz9JhRA7P3HC0OmIg47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFVqIOOwxEKY/BwUmvv0yJlvuSQnrkHk
ZJuTTKSVmRt4UrhVchSPog3D+aWmGArI/l4km9vJcRd7whK2AQhYGkgKIII6XAhjqdV8wjucSrT2dF8dRAgeFJfWNbl
QI1MSikduEiQIARIgZpS2gkWPSavVjFTWXvfNU17Ei9gSPQpic7ht6HAMkp4iaAgCEhSPog3D+aWmGArI/l4km9vJcR
d7whoMCMGHma4GEOSG7sACIkAqkIYVNm9GQkMLlJ5XUTipuLFF7avWjDmerGU2dh/Plf9cHzb6Wc4W7FLTISu6fGSXb
pGBt5Un+OZ72a6PhWYHEn4KPAoUj6INw/mlphgKyP5eJJvbyXEXe8ISIgogAX0bC5lAVQaBv6HoRJxaRdR6Bd7QbPRO
uYGmognE/xMYARI8ChSPog3D+aWmGArI/l4km9vJcRd7whIiCiABfRsLmUBVBoG/oehEnFpF1HoF3tBs9E65gaaiCcT
/ExgBGAEaAhBXIn4KPAoUj6INw/mlphgKyP5eJJvbyXEXe8ISIgogAX0bC5lAVQaBv6HoRJxaRdR6Bd7QbPROuYGmog
nE/xMYARI8ChSPog3D+aWmGArI/l4km9vJcRd7whIiCiABfRsLmUBVBoG/oehEnFpF1HoF3tBs9E65gaaiCcT/ExgBG
AEaLWNvc21vczEyaHBtenZ1Zmt3Znh1NWp5azk1MmE0YTk4dWVubHozZ2UzanF0YRI3aGVybWVzIDEuOC4wKzkxZmY4
ZmJhIChodHRwczovL2hlcm1lcy5pbmZvcm1hbC5zeXN0ZW1zKRJmClAKRgofL2Nvc21vcy5jcnlwdG8uc2VjcDI1Nms
xLlB1YktleRIjCiECFY9es0STPxeb35klZdCYTaASlCMvzaaIM26KkO5OLA0SBAoCCAEYAhISCgwKBXN0YWtlEgMxMD
kQ684GGkBfTk7vKfa9w6SOd3IieT3QIcg5m/5VVpEDcMaccLnliiZcucQ1tTuhwKJbzRvXeVFbQor2zPFzi46/C1NMg
K+v' | base64 --decode | ~/go/bin/protoscope
1: {
  1: {
    1: {"/ibc.core.client.v1.MsgUpdateClient"}
    2: {
      1: {"07-tendermint-0"}
      2: {
        1: {"/ibc.lightclients.tendermint.v1.Header"}
[…SNIP…]
      3: {"cosmos12hpmzvufkwfxu5jyk952a4a98uenlz3ge3jqta"}
    }
  }
  2: {"hermes 1.8.0+91ff8fba (https://hermes.informal.systems)"}
}
2: {
  1: {
    1: {
      1: {"/cosmos.crypto.secp256k1.PubKey"}
      2: {1: {`02158f5eb344933f179bdf992565d0984da01294232fcda688336e8a90ee4e2c0d`}}
    }
    2: {1: {1: 1}}
    3: 2
  }
  2: {
    1: {
      1: {"stake"}
      2: {"109"}
    }
    2: 108395
  }
}
3: {
  `5f4e4eef29f6bdc3a48e777222793dd021c8399bfe5556910370c69c70b9e58a265cb9c435b53ba1`
  `c0a25bcd1bd779515b428af6ccf1738b8ebf0b534c80afaf`
}
bperry@bperry-ThinkPad-X1-Carbon-6th:~$
```

Example Decoded Protobuf Packet and Details

Atredis also experienced signing and other verification errors during attacks when altering `protobuf` packets over-the-wire, ensuring that the cross-chain key signing and other checks were implemented and verified correctly.

```
HTTP/1.1 200 OK

Content-Type: application/json

X-Server-Time: 1708701716

Date: Fri, 23 Feb 2024 15:21:56 GMT

Content-Length: 319

Connection: close



{"jsonrpc":"2.0","id":"e92f40e9-48d9-41f5-90db-
fe3f70ee2124","result":{"code":9,"data":"","log":"fee payer address:
\ufffd3L\ufffd(\ufffd\t\ufffd\ufffd\ufffd;\ufffd\ufffd\ufffd\ufffdV\ufffd% does not exist:
unknown
address","codespace":"sdk","hash":"EDB2C18D38F1AB405C7A27A3F8B486CF28E76F9D2EE77C57630B218A
188D59A4"}}
```

Using Unknown Address

```
HTTP/1.1 200 OK

Content-Type: application/json

X-Server-Time: 1708702188

Date: Fri, 23 Feb 2024 15:29:48 GMT

Content-Length: 314

Connection: close



{"jsonrpc":"2.0","id":"faa4f576-c11a-47f4-b16c-
c699fc0d4aef","result":{"code":5,"data":"","log":"string could not be parsed as address:
decoding bech32 failed: invalid character not part of charset: 105: invalid
address","codespace":"ibc","hash":"F5840EC4C38B53F7249957246582C67A07AC52B5813B8800BE715098
66F51759"}}
```

Using Invalid Address

Channel Upgrade Process and Flow

Several states exist during a channel upgrade. Before the upgrade happens and after the upgrade is successful, the channel state is `OPEN`.

```
$ simd q ibc channel channels --node http://localhost:27000
channels:
- channel_id: channel-0
  connection_hops:
  - connection-0
  counterparty:
    channel_id: channel-0
    port_id: transfer
  ordering: ORDER_UNORDERED
  port_id: transfer
  state: STATE_OPEN
  upgrade_sequence: "0"
  version: ics20-1
height:
  revision_height: "127"
  revision_number: "0"
pagination:
  next_key: null
  total: "0"
```

Channel in Open State

Once a vote passes on both chains, a `ChanUpgradeTry` packet is sent by the relayer to each chain's RPC. If the chains verify the proposed timeout for the upgrade, the channel state is switched to `FLUSHING`. Atredis found no way to put the channel into the `FLUSHING` state inadvertently.

Atredis also found no way to intentionally put the channel in the `FLUSHING` state and keep it there. By holding subsequent upgrade packets in the attack proxy during the upgrade process, Atredis was able to ensure timeouts during the upgrade process resulted in a rollback of any changes to the chain's communication with the channel. By holding key requests or dropping them during upgrades, handshake timeout thresholds were met, rolling back any changes. Atredis also ensured no new packets were processed when the channel was in a `FLUSHING` state. While upgrade packets were held and the state was in `FLUSHING`, Atredis attempted to send packets asynchronously which were not operated on. Only packets sent prior to the state change were processed.

Once the `FLUSHING` state is over, ensuring any old version packets had been processed before the upgrade expects different parameters, the channel moves to the `FLUSHCOMPLETE` state and the relayer responds to the chains with a `ChanUpgradeConfirm` packet. Once both ends respond, the state of the channel is moved to `OPEN`, and a `ChanUpgradeOpen` packet is sent from the relayer to the chains. A `ChanUpgradeCancel` packet can be sent during the upgrade by either chain to the relayer which will cancel the upgrade for both chains, reverting the channel back to the previous version.

To ensure the chains are agreeing to the same upgrade version, the proposal specifies a JSON stanza as a version string. These versions are verified by each chain to ensure they match.

```
195   // NOTE: if an upgrade exists (crossing hellos) then use existing upgrade fields
196   // otherwise, run the upgrade init sub-protocol
197   if isCrossingHello {
198     proposedUpgradeFields = upgrade.Fields
199   } else {
200     // NOTE: OnChanUpgradeInit will not be executed by the application
201     upgrade, err = k.ChanUpgradeInit(ctx, portID, channelID, proposedUpgradeFields)
202     if err != nil {
203       return types.Channel{}, types.Upgrade{}, errorsmod.Wrap(err, "failed to initialize upgrade")
204     }
205
206     channel, upgrade = k.WriteUpgradeInitChannel(ctx, portID, channelID, upgrade, upgrade.Fields.Version)
207   }
208
209   if err := k.checkForUpgradeCompatibility(ctx, proposedUpgradeFields, counterpartyUpgradeFields); err != nil {
210     return types.Channel{}, types.Upgrade{}, errorsmod.Wrap(err, "failed upgrade compatibility check")
211   }
212
213   // if the counterparty sequence is greater than the current sequence, we fast-forward to the counterparty sequence.
214   if counterpartyUpgradeSequence > channel.UpgradeSequence {
215     channel.UpgradeSequence = counterpartyUpgradeSequence
216     k.SetChannel(ctx, portID, channelID, channel)
217   }
218
219   if err := k.startFlushing(ctx, portID, channelID, &upgrade); err != nil {
220     return types.Channel{}, types.Upgrade{}, err
221   }
```

Line 209 in upgrade.go Verifying Upgrade Compatibility

```
{
  "title": "Channel upgrade init",
  "summary": "Channel upgrade init",
  "messages": [
  {
         "@type": "/ibc.core.channel.v1.MsgChannelUpgradeInit",
         "signer": "cosmos10d07y265gmmuvt4z0w9aw880jnsr700j6zn9kn",
         "port_id": "transfer",
         "channel_id": "channel-0",
         "fields": {
             "ordering": "ORDER_UNORDERED",
             "connection_hops": ["connection-0"],
             "version": "{\"fee_version\":\"ics29-1\",\"app_version\":\"ics20-1\"}"
          }
   }
   ],
   "metadata": "AQ==",
   "deposit": "100005stake"
}
```

Channel Upgrade Proposal with JSON Version String

Atredis found no way to create malicious JSON as a version that would cause different versions to be interpreted by different chains, leading to a broken or inconsistent channel upgrade. One possible issue Atredis verified during manual code review was the seemingly unsafe usage of unmarshaling the JSON version string.

```
func MetadataFromVersion(version string) (Metadata, error)
{
    var metadata Metadata err := ModuleCdc.UnmarshalJSON([]byte(version), &metadata)
    if err != nil
    {
        return Metadata{}, errorsmod.Wrapf(ErrInvalidVersion, "failed to unmarshal metadata
from version: %s", version)
    }
    return metadata, nil
}
```

The UnmarshalJSON Routine Being Called

This ultimately ends up calling the standard Go library routine `json.Unmarshal()`.

```
if err := json.Unmarshal(bz, &alias); err != nil {
    return err
}
```

Calling json.Unmarshal() on Version String from Proposal

However, Atredis confirmed that this current implementation does not introduce a security risk and could not achieve any kind of deserialization or remote code execution. After a successful upgrade, the version string will change to the string defined in the original proposal.

```
$ simd q ibc channel channels --node http://localhost:27000
channels:
- channel_id: channel-0
  connection_hops:
  - connection-0
  counterparty:
    channel_id: channel-0
    port_id: transfer
  ordering: ORDER_UNORDERED
  port_id: transfer
  state: STATE_OPEN
  upgrade_sequence: "0"
  version: ics20-1
height:
  revision_height: "127"
  revision_number: "0"
pagination:
  next_key: null
  total: "0"
```

Version String Pre-Upgrade

```
$ simd q ibc channel channels --node http://localhost:27000
channels:
- channel_id: channel-0
  connection_hops:
  - connection-0
  counterparty:
    channel_id: channel-0
    port_id: transfer
  ordering: ORDER_UNORDERED
  port_id: transfer
  state: STATE_OPEN
  upgrade_sequence: "1"
  version: '{"fee_version":"ics29-1","app_version":"ics20-1"}'
height:
  revision_height: "165"
  revision_number: "0"
pagination:
  next_key: null
  total: "0"
```

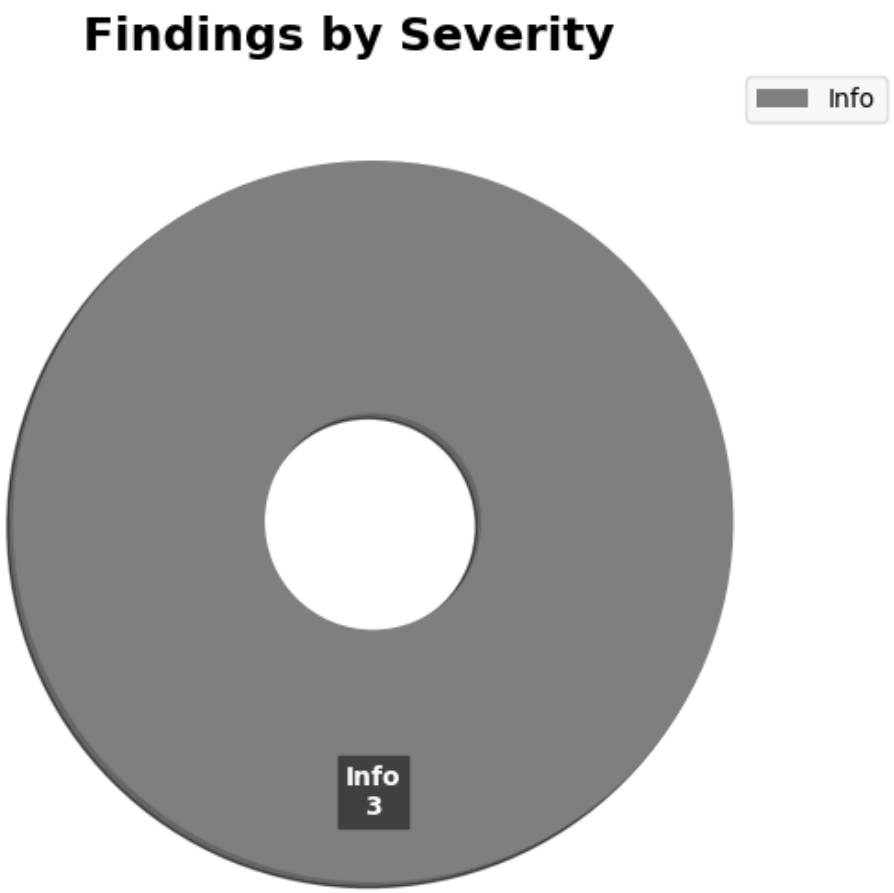
Successful Upgrade Version String

Findings Summary

In performing testing for this assessment, Atredis Partners identified **three (3) informational** findings. No high or critical severity findings were noted.

Atredis defines vulnerability severity ranking as follows:

- **Critical:** This vulnerability rating is reserved for the most impactful vulnerabilities to high criticality infrastructure or components. They have a high likelihood of exploitation and, once exploited, have a large negative impact for the target under review.
- **High:** Exploitation of high-rated vulnerabilities result in compromise of sensitive, protected areas of the target, but have mitigating factors reducing the severity from Critical to High. These mitigating factors typically relate to limitations on total possible impact, requirements to chain multiple issues together to achieve an attacker goal, or access requirements that aren't easily met.
- **Medium:** Exploitation of Medium severity vulnerabilities does not, by itself, provide anything of direct use to the attacker. It may disclose information useful in exploitation of a High-rated vulnerability, or may be chained together with multiple Medium or Low-rated vulnerabilities to collectively provide use to an attacker, such as a new avenue for attack or new privilege level with little obvious and direct benefit to the attacker.
- **Low:** Vulnerabilities given a Low designation relate to minor reductions in overall defensiveness or otherwise provide minor benefits to attackers. These often include weaknesses that may feasibly become exploitable in the future but present little risk, by themselves, to the current environment.



Remediation Tasks

In the case of this assessment, remediation tasks are not complex, and relate to configuration changes and general hardening practices. In remediating most findings, minimal (if any) changes to application source code are required, and no coordination with vendors or third parties is necessary to address findings.

As described elsewhere, no directly or indirectly exploitable conditions were noted from either an authenticated or unauthenticated user perspective, and as such, remediation tasks would be primarily driven by the objective of increasing overall application hardening and diminishing the impact of any potential future vulnerabilities that may arise in the platform.

Findings and Recommendations

The following section outlines findings identified via manual and automated testing over the course of this engagement. Where necessary, specific artifacts to validate or replicate issues are included, as well as Atredis Partners' views on finding severity and recommended remediation.

Findings Summary

The below tables summarize the number and severity of the unique issues identified throughout the engagement.

CRITICAL	HIGH	MEDIUM	LOW	INFO
0	0	0	0	3

Findings Detail

FINDING NAME	SEVERITY
RPC Communication over Plaintext HTTP	Info
Possible Out-of-Bounds Slice Access	Info
Deprecated Protobuf Package in Use	Info

RPC Communication over Plaintext HTTP

Severity: Info

Finding Overview

During testing, Atredis noticed that default RPC communications for the Hermes relayer were over plaintext HTTP. This may allow a local attacker to read or alter data via man-in-the-middle attacks.

Finding Detail

By default, the setup instructions for Hermes to operate include plaintext local HTTP and Web Socket RPCs.

```
[[chains]]
id = 'chain1'
type = 'CosmosSdk'
rpc_addr = 'http://localhost:27000'
grpc_addr = 'http://localhost:27002'
event_source = { mode = 'push', url = 'ws://localhost:27000/websocket', batch_delay =
'200ms' }
rpc_timeout = '15s'
trusted_node = true
account_prefix = 'cosmos'
key_name = 'wallet'
store_prefix = 'ibc'
gas_price = { price = 0.001, denom = 'stake' }
gas_multiplier = 1.2
default_gas = 1000000
max_gas = 10000000
max_msg_num = 30
max_tx_size = 2097152
clock_drift = '5s'
max_block_time = '30s'
trusting_period = '14days'
trust_threshold = { numerator = '2', denominator = '3' }

[[chains]]
id = 'chain2'
type = 'CosmosSdk'
rpc_addr = 'http://localhost:27010'
grpc_addr = 'http://localhost:27012'
event_source = { mode = 'push', url = 'ws://localhost:27010/websocket', batch_delay =
'200ms' }
rpc_timeout = '15s'
trusted_node = true
account_prefix = 'cosmos'
key_name = 'wallet'
store_prefix = 'ibc'
gas_price = { price = 0.001, denom = 'stake' }
gas_multiplier = 1.2
default_gas = 1000000
max_gas = 10000000
max_msg_num = 30
max_tx_size = 2097152
clock_drift = '5s'
max_block_time = '30s'
trusting_period = '14days'
trust_threshold = { numerator = '2', denominator = '3' }
```

Default Hermes Configuration

Atredis could not find a real-world attack that would cause this issue to become a vulnerability. In fact, some systems, such as Debian's package manager `apt`, still use HTTP because it provides separate signing verification. In the case of the default instructions for Hermes, similar signing using the keys of the two separate chains ensures that tampering over the wire is generally prevented.

To add to that, the only communication over the channel between the two chains should be public information by definition. Plaintext HTTP in this case should not leak any potentially private information unless a relayer or chain RPC has been misconfigured somehow.

Recommendation(s)

Either update documentation with instructions for setting up SSL-enabled RPC communications or make a note in the documentation that RPC communication in production should be implemented over HTTPS instead of the simpler HTTP.

References

CWE-319: Plaintext Transmission of Sensitive Information

https://cwe.mitre.org/data/definitions/319.html

Possible Out-of-Bounds Slice Access

Severity: Info

Finding Overview

While performing static code analysis of the channel upgrade code, Atredis identified several occurrences where an array element is directly accessed without validating the length first. In general, if an application attempts to grab a value from an empty array, the application will panic and shut down. Atredis would note that the reviewed application should not crash as it leverages the Go standard library to prevent from shutting down the process. Atredis was unable to identify an exploit path during testing.

Finding Detail

During static code analysis Atredis noticed the slice `ConnectionHops` was directly accessed at index 0 in many function calls, such as this example in `ChanUpgradeTry()`.

```
connection, found := k.connectionKeeper.GetConnection(ctx, channel.ConnectionHops[0])
```

modules/core/04-channel/keeper/upgrade.go:96

However, after performing more code review and speaking with the team, the following functions `ValidateBasic()` are called prior to our suspect line of code during normal application execution.

The first function is used when attempting to perform a `MsgChannelUpgradeInit`.

```
// ValidateBasic performs a basic validation of the proposed upgrade fields
func (uf UpgradeFields) ValidateBasic() error {
    if !slices.Contains(connectiontypes.SupportedOrderings, uf.Ordering.String()) {
        return errorsmod.Wrap(ErrInvalidChannelOrdering, uf.Ordering.String())
    }

    if len(uf.ConnectionHops) != 1 {
        return errorsmod.Wrap(ErrTooManyConnectionHops, "current IBC version only
supports one connection hop")
    }

    if strings.TrimSpace(uf.Version) == "" {
        return errorsmod.Wrap(ErrInvalidChannelVersion, "version cannot be empty")
    }

    return nil
}
```

modules/core/04-channel/types/upgrade.go:46

The following function is used when a `MsgChannelUpgradeTry()` is called.

```
func (msg MsgChannelUpgradeTry) ValidateBasic() error {
    if err := host.PortIdentifierValidator(msg.PortId); err != nil {
        return errorsmod.Wrap(err, "invalid port ID")
    }

    if !IsValidChannelID(msg.ChannelId) {
        return ErrInvalidChannelIdentifier
    }

    if len(msg.ProposedUpgradeConnectionHops) == 0 {
        return errorsmod.Wrap(ErrInvalidUpgrade, "proposed connection hops cannot be
empty")
    }
```

modules/core/04-channel/types/msgs.go:459

These `ValidateBasic` functions do check the `ConnectionHops` length and validates the actual length of the slice is greater than zero. These prevent the possible access of a non-existent array slice. However, it does not prevent future code from calling the `ChanUpgradeTry` function accessing array slices unsafely directly and possibly before `ValidateBasic` or `MsgChannelUpgradeTry`.

Recommendation(s)

Atredis would recommend adding length validation of the `ConnectionHops` slice inside any the functions where the index is accessed directly. The example noted in the issue is in the `ChanUpgradeTry` function but the pattern was noted in other places as well.

References

CWE-129: Improper Validation of Array Index:
https://cwe.mitre.org/data/definitions/129.html

Deprecated Protobuf Package in Use

Severity: Info

Finding Overview

The Go module `github.com/golang/protobuf` has been superseded by the `google.golang.org/protobuf` module. No security issues are present because of this, but is more a code quality recommendation.

Finding Detail

During the code review process of the engagement, Atredis noted the use of the package `github.com/golang/protobuf` in the go.mod file.

```
require (
---Snipped---
    github.com/cosmos/ics23/go v0.10.0
    github.com/golang/protobuf v1.5.3
    github.com/grpc-ecosystem/grpc-gateway v1.16.0
    github.com/hashicorp/go-metrics v0.5.3
    github.com/spf13/cast v1.6.0
    github.com/spf13/cobra v1.8.0
    github.com/spf13/viper v1.18.2
    github.com/stretchr/testify v1.8.4
    google.golang.org/genproto/googleapis/api v0.0.0-20231120223509-83a465c0220f
    google.golang.org/grpc v1.61.1
    google.golang.org/protobuf v1.32.0
    gopkg.in/yaml.v2 v2.4.0
)
```

go.mod Contents

Recommendation(s)

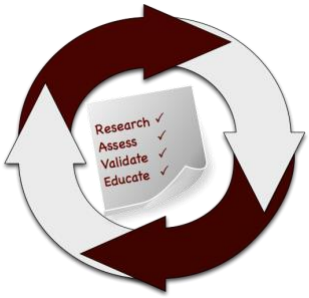
As the `go.mod` file already utilizes the `google.golang.org/protobuf` module, Atredis recommends removing the superseded package.

References

CWE-1104: Use of Unmaintained Third Party Components:
https://cwe.mitre.org/data/definitions/1104.html

Github Issue:
https://github.com/golang/protobuf/pull/1306

Appendix I: Assessment Methodology



Atredis Partners draws on our extensive experience in penetration testing, reverse engineering, hardware/software exploitation, and embedded systems design to tailor each assessment to the specific targets, attacker profile, and threat scenarios relevant to our client's business drivers and agreed upon rules of engagement.

Where applicable, we also draw on and reference specific industry best practices, regulations, and principles of sound systems and software design to help our clients improve their products while simultaneously making them more stable and secure.

Our team takes guidance from industry-wide standards and practices such as the National Institute of Standards and Technology's (NIST) Special Publications, the Open Web Application Security Project (OWASP), and the Center for Internet Security (CIS).

Throughout the engagement, we communicate findings as they are identified and validated, and schedule ongoing engagement meetings and touchpoints, keeping our process open and transparent and working closely with our clients to focus testing efforts where they provide the most value.

In most engagements, our primary focus is on creating purpose-built test suites and toolchains to evaluate the target, but we do utilize off-the-shelf tools where applicable as well, both for general patch audit and best practice validation as well as to ensure a comprehensive and consistent baseline is obtained.

Research and Profiling Phase

Our research-driven approach to testing begins with a detailed examination of the target, where we model the behavior of the application, network, and software components in their default state. We map out hosts and network services, patch levels, and application versions. We frequently use a number of private and public data sources to collect Open Source Intelligence about the target, and collaborate with client personnel to further inform our testing objectives.

For network and web application assessments, we perform network and host discovery as well as map out all available application interfaces and inputs. For hardware assessments, we study the design and implementation, down to a circuit-debugging level. In reviewing source code or compiled application code, we map out application flow and call trees and develop a solid working understand of how the application behaves, thus helping focus our validation and testing efforts on areas where vulnerabilities might have the highest impact to the application's security or integrity.

Analysis and Instrumentation Phase

Once we have developed a thorough understanding of the target, we use a number of specialized and custom-developed tools to perform vulnerability discovery as well as binary, protocol, and runtime analysis, frequently creating engagement-specific software tools which we share with our clients at the close of any engagement.

We identify and implement means to monitor and instrument the behavior of the target, utilizing debugging, decompilation and runtime analysis, as well as making use of memory and filesystem forensics analysis to create a comprehensive attack modeling testbed. Where they exist, we also use common off-the-shelf, open-source and any extant vendor-proprietary tools to aid in testing and evaluation.

Validation and Attack Phase

Using our understanding of the target, our team creates a series of highly-specific attack and fault injection test cases and scenarios. Our selection of test cases and testing viewpoints are based on our understanding of which approaches are most relevant to the target and will gain results in the most efficient manner, and built in collaboration with our client during the engagement.

Once our test cases are validated and specific attacks are confirmed, we create proof-of-concept artifacts and pursue confirmed attacks to identify extent of potential damage, risk to the environment, and reliability of each attack scenario. We also gather all the necessary data to confirm vulnerabilities identified and work to identify and document specific root causes and all relevant instances in software, hardware, or firmware where a given issue exists.

Education and Evidentiary Phase

At the conclusion of active testing, our team gathers all raw data, relevant custom toolchains, and applicable testing artifacts, parses and normalizes these results, and presents an initial findings brief to our clients, so that remediation can begin while a more formal document is created. Additionally, our team shares confirmed high-risk findings throughout the engagement so that our clients may begin to address any critical issues as soon as they are identified.

After the outbrief and initial findings review, we develop a detailed research deliverable report that provides not only our findings and recommendations but also an open and transparent narrative about our testing process, observations and specific challenges in developing attacks against our targets, from the real world perspective of a skilled, motivated attacker.

Automation and Off-The-Shelf Tools

Where applicable or useful, our team does utilize licensed and open-source software to aid us throughout the evaluation process. These tools and their output are considered secondary to manual human analysis, but nonetheless provide a valuable secondary source of data, after careful validation and reduction of false positives.

For runtime analysis and debugging, we rely extensively on Hopper, IDA Pro and Hex-Rays, as well as platform-specific runtime debuggers, and develop fuzzing, memory analysis, and other testing tools primarily in Ruby and Python.

In source auditing, we typically work in Visual Studio, Xcode and Eclipse IDE, as well as other markup tools. For automated source code analysis we will typically use the most appropriate toolchain for the target, unless client preference dictates another tool.

Network discovery and exploitation make use of Nessus, Metasploit, and other open-source scanning tools, again deferring to client preference where applicable. Web application runtime analysis relies extensively on the Burp Suite, Fuzzer and Scanner, as well as purpose-built automation tools built in Go, Ruby and Python.

Engagement Deliverables

Atredis Partners deliverables include a detailed overview of testing steps and testing dates, as well as our understanding of the specific risk profile developed from performing the objectives of the given engagement.

In the engagement summary we focus on "big picture" recommendations and a high-level overview of shared attributes of vulnerabilities identified and organizational-level recommendations that might address these findings.

In the findings section of the document, we provide detailed information about vulnerabilities identified, provide relevant steps and proof-of-concept code to replicate these findings, and our recommended approach to remediate the issues, developing these recommendations collaboratively with our clients before finalization of the document.

Our team typically makes use of both DREAD and NIST CVE for risk scoring and naming, but as part of our charter as a client-driven and collaborative consultancy, we can vary our scoring model to a given client's preferred risk model, and in many cases will create our findings using the client's internal findings templates, if requested.

Sample deliverables can be provided upon request, but due to the highly specific and confidential nature of Atredis Partners' work, these deliverables will be heavily sanitized, and give only a very general sense of the document structure.

Appendix II: Engagement Team Biographies

Shawn Moyer, Founding Partner and CEO

Shawn Moyer scopes, plans, and coordinates security research and consulting projects for the Atredis Partners team, including reverse engineering, binary analysis, advanced penetration testing, and private vulnerability research. As CEO, Shawn works with the Atredis leadership team to build and grow the Atredis culture, making Atredis Partners a home for some of the best minds in information security, and ensuring Atredis continues to deliver research and consulting services that exceed our client's expectations.

Experience

Shawn brings over 25 years of experience in information security, with an extensive background in penetration testing, advanced security research including extensive work in mobile and Smart Grid security, as well as advanced threat modeling and embedded reverse engineering.

Shawn has served as a team lead and consultant in enterprise security for numerous large initiatives in the financial sector and the federal government, including IBM Internet Security Systems' X-Force, MasterCard, a large Federal agency, and Wells Fargo Securities, all focusing on emerging network and application attacks and defenses.

In 2010, Shawn created Accuvant Labs' Applied Research practice, delivering advanced research-driven consulting to numerous clients on mobile platforms, critical infrastructure, medical devices and countless other targets, growing the practice 1800% in its first year.

Prior to Accuvant, Shawn helped develop FishNet Security's penetration testing team as a principal security consultant, growing red team offerings and advanced penetration testing services, while being twice selected as a consulting MVP.

Key Accomplishments

Shawn has written on emerging threats and other topics for Information Security Magazine and ZDNet, and his research has been featured in the Washington Post, BusinessWeek, NPR and the New York Times. Shawn is a twelve-time speaker at the Black Hat Briefings and has been an invited speaker at other notable security conferences around the world.

Shawn is likely best known for delivering the first public research on social network security, pointing out much of the threat landscape still exists on social network platforms today. Shawn also co-authored an analysis of the state of the art in web browser exploit mitigation, creating the first in-depth comparison of browser security models along with Dr. Charlie Miller, Chris Valasek, Ryan Smith, Joshua Drake, and Paul Mehta.

Shawn studied Computer and Network Information Systems at Missouri University and the University of Louisiana at Lafayette, holds numerous information security certifications, and has been a frequent presenter at national and international security industry conferences.

Brandon Perry, Principal Research Consultant

Brandon Perry's responsibilities include complex web application and web services assessments, software reverse engineering, and source code reviews as well as red team and attack simulation engagements.

Experience

Most recently, Brandon was a Senior Penetration Tester at NTT Security, and contributed to the Foxglove Security blog at http://www.foxglovesecurity.com along with teammates (and now fellow Atredians) Justin Kennedy and Stephen Breen.

Prior to NTT Security, Brandon worked as a developer at Rapid7, supporting the Metasploit Framework and Nexpose vulnerability scanner. Brandon has also worked as a security engineer for Bethesda, working on AAA game titles such as DOOM, Fallout 4, and Elder Scrolls Online.

Brandon's extensive experience as a software engineer across several complex application stacks gives him key, practitioner-level insight into application security from the big picture down to the specifics of implementation and remediation. This development background makes Brandon uniquely suited for roles where interfacing and collaborating with development teams is crucial to a successful engagement.

Key Accomplishments

Brandon is the author of the No Starch book "Gray Hat C#", and the co-author of "Wicked Cool Shell Scripts, 2nd edition", and has spoken at DerbyCon and Infosec Southwest on web application vulnerabilities and exploit writing. Brandon has also heavily contributed to the Metasploit Framework with code and exploits, as well as several other open source security tools.

James Cook, Principal Research Consultant

James executes highly technical software security, network, and web application assessments and advanced red team assessments.

Experience

James has over 6 years of experience in the information security industry. James has developed and released numerous open-source security tools that are used by many information security professionals. He holds the Offensive Security Certified Professional (OSCP) certification and has performed a wide variety of security assessments including network penetration testing, application security assessments, full-scope red team engagements, adversarial simulation, and physical penetration testing

Prior to joining Atredis Partners, James performed network penetration tests as a Senior Security Consultant on Optiv's Attack and Penetration team.

Key Accomplishments

James has presented at Black Hat Arsenal and DerbyCon. James has also contributed to the Metasploit Framework and Veil Evasion Toolkit, as well as several other open source security tools.

Sara Bettes, Client Operations Associate

Sara Bettes assists the creation and completion of projects at Atredis Partners, ranging from the full pre-sales process to project design and management, to final delivery and follow-up. Her goals are to ensure all projects are executed in a way that reaches the goals of the client and assists the consultants at every turn.

Experience

Prior to joining Atredis Partners, Sara led a team that planned international sporting competitions, Olympic and national team qualifying events, as well as supported the mission of multiple non-profits. Her experience includes Live Sports Commentating, Staffing Management, Safety Plan Creation, Event Development, Public Relations, and Marketing efforts.

Key Accomplishments

Sara earned a bachelor's degree in Mass Communications with an emphasis in Broadcast and Public Relations from Oklahoma City University.

Ryan Hanson, Research Science Director

Ryan has over ten years of professional experience in the software industry. Prior to entering the security industry, Ryan was a software engineer with an emphasis on infrastructure and application security. He managed and worked with a team of engineers to build, deploy, and secure cross-platform applications.

Experience

Ryan has developed and released numerous open-source security tools that are used by many information security professionals. He holds the Offensive Security Certified Professional (OSCP) certification and has performed a wide variety of security assessments. Ryan's strengths shine when simulating advanced threats while conducting red team exercises. Using a combination of reversing engineering and logical network attacks, he has crafted creative attack chains to compromise many organization's high value assets. Ryan's in-depth reporting of these attack chains has helped Fortune 100 companies identify and mitigate weaknesses in their overall security posture.

Key Accomplishments

Ryan's strong background in software engineering has been a contributing factor to his success in the security industry. His deep understanding of software has aided him in discovering several critical vulnerabilities in Microsoft Office products. Ryan has received industry recognition for these findings and was awarded with the 2017 Pwnie Award for Best Client-Side Bug.

Molly Vukusich, Client Operations Associate

Molly Vukusich supports nearly every phase of the project lifecycle at Atredis Partners, from pre-sales, to project planning and management, to project delivery, readout and follow-up. She aims to increase efficiency of project execution and client communication for the benefit of both the consultants and clients.

Experience

Molly has over 14 years of experience in marketing and project management roles in various industries such as Healthcare, Finance, Sports & Recreation, and Non-Profit. Her experience includes copywriting and editing (both technical and promotional), creative strategy development, data analysis, event planning, graphic design, and website management.

Key Accomplishments

Molly earned a bachelor's degree in Mass Communications with an emphasis in Advertising and Public Relations from Oklahoma City University.

Appendix III: About Atredis Partners

Atredis Partners was created in 2013 by a team of security industry veterans who wanted to prioritize offering quality and client needs over the pressure to grow rapidly at the expense of delivery and execution. We wanted to build something better, for the long haul.

In six years, Atredis Partners has doubled in size annually, and has been named three times to the Saint Louis Business Journal's "Fifty Fastest Growing Companies" and "Ten Fastest Growing Tech Companies". Consecutively for the past three years, Atredis Partners has been listed on the Inc. 5,000 list of fastest growing private companies in the United States.

The Atredis team is made up of some of the greatest minds in Information Security research and penetration testing, and we've built our business on a reputation for delivering deeper, more advanced assessments than any other firm in our industry.

Atredis Partners team members have presented research over forty times at the BlackHat Briefings conference in Europe, Japan, and the United States, as well as many other notable security conferences, including RSA, ShmooCon, DerbyCon, BSides, and PacSec/CanSec. Most of our team hold one or more advanced degrees in Computer Science or engineering, as well as many other industry certifications and designations. Atredis team members have authored several books, including *The Android Hacker's Handbook*, *The iOS Hacker's Handbook*, *Wicked Cool Shell Scripts*, *Gray Hat C#*, and *Black Hat Go*.

While our client base is by definition confidential and we often operate under strict nondisclosure agreements, Atredis Partners has delivered notable public security research on improving the security at Google, Microsoft, The Linux Foundation, Motorola, Samsung and HTC products, and were the first security research firm to be named in Qualcomm's Product Security Hall of Fame. We've received four research grants from the Defense Advanced Research Project Agency (DARPA), participated in research for the CNCF (Cloud Native Computing Foundation) to advance the security of Kubernetes, worked with OSTIF (The Open Source Technology Improvement Fund) and The Linux Foundation on the Core Infrastructure Initiative to improve the security and safety of the Linux Kernel, and have identified entirely new classes of vulnerabilities in hardware, software, and the infrastructure of the World Wide Web.

In 2015, we expanded our services portfolio to include a wide range of advanced risk and security program management consulting, expanding our services reach to extend from the technical trenches into the boardroom. The Atredis Risk and Advisory team has extensive experience building mature security programs, performing risk and readiness assessments, and serving as trusted partners to our clients to ensure the right people are making informed decisions about risk and risk management.