Interchain Berlin, Interchain Accounts Module

Security Assessment

December 17, 2021

Prepared for:
Sean King
Interchain Berlin

Prepared by:
Dominik Czarnota
Alex Useche

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Classification and Copyright

This report is confidential and intended for the sole internal use of Interchain Berlin.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Table of Contents

Executive Summary

Overview

Interchain Berlin engaged Trail of Bits to review the security of its Inter-Blockchain Communication (IBC) protocol's interchain accounts module. From December 6 to December 17, 2021, a team of two consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

We focused our testing efforts on the identification of flaws that could result in a compromise or lapse of confidentiality, integrity, or availability of the target system. We performed automated testing and a manual review of the code, in addition to reviewing the specification and running tests and a demo application.

Summary of Findings

During the audit, we discovered a high-severity issue (TOB-IBCICA-5) involving the possibility that untrusted cosmos transactions, when deserialized, could cause application crashes and denials of service. In addition, we uncovered one low-severity and three informational-severity issues regarding data validation and auditing and logging mechanisms. Lastly, we identified four issues of undetermined severity that should be addressed, as they could lead to vulnerabilities of significant impact. A summary of the findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Low	1
Informational	3
Undetermined	4

CATEGORY BREAKDOWN

Category	*Count*
Auditing and Logging	3
Data Validation	4
Patching	1
Timing	1

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager dan@trailofbits.com	**Mary O'Brien**, Project Manager mary.obrien@trailofbits.com

The following engineers were associated with this project:

Dominik Czarnota, Consultant dominik.czarnota@trailofbits.com	**Alex Useche**, Consultant alex.useche@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
December 6, 2021	Pre-project kickoff call
December 10, 2021	Status update meeting #1
December 17, 2021	Final meeting
January 3, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the Interchain Berlin IBC interchain accounts module. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the software development kit (SDK) perform data validation?
- Are errors handled properly throughout the codebase?
- Does the codebase contain flawed logic that could cause panics, facilitating denial-of-service attacks?
- Could a malicious controller chain take advantage of a host chain?
- Does the codebase reflect the documented desired properties?
- Does the version negotiation logic contain flaws that would allow deviations from specification requirements?
- Is data safely serialized and deserialized?

Project Targets

The engagement involved a review and testing of the targets listed below.

`ibc-go`

Repository	https://github.com/cosmos/ibc-go/
Version	`22e87deacd35f287586d1e2d529e869ce1cfa208`
Type	Go

`ics-027-interchain-accounts`

Repository	https://github.com/cosmos/ibc/spec/app/ics-027-interchain-accounts
Version	`7046202b645c65b1a2b7f293312bca5d651a13a4`
Type	Specification

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- **`ics-027-interchain-accounts`**: We reviewed the specification and the implementation of the account manager standard as defined in the IBC documentation. We also conducted research on topics like channel creation, account registration, version negotiation, and controlling of the logic flow.
- **Handshake**: We checked the handshake logic for error handling, data validation, and correctness issues.
- **Keeper**: We verified message validation and execution.
- **gRPC**: We checked the logic responsible for handling gRPC requests for issues that could result from insufficient data validation.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of certain system elements, which may warrant further review. The following is a summary of the coverage limitations of the engagement:

- The IBC-Go protocol is fairly complex. Issues of undetermined severity might require additional testing and verification to ascertain their impact on the system.
- Our ability to test dynamically was limited. While we conducted some dynamic testing on the `interchain-accounts` project, additional components such as the handshake process and the relayer logic could benefit from further focused testing.
- We could not properly test the solution against its functionality in practice, with a proper authentication module and multiple parties and channels. A proper authentication module was out of scope of this audit; we recommend conducting an additional review of the project to test for cases in which an authentication module is introduced to a blockchain using the interchain accounts module.
- The solution would also benefit from testing of the entire system's logic with multiple controller chains targeting a given host chain, as the current test cases test only situations in which a single controller chain connects to a single host chain.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Outdated and vulnerable dependencies	Patching	**Informational**
2	Revision and client identifier regex formats accept newlines, which may be unintended behavior	Data Validation	**Undetermined**
3	The IsValidAddr regex accepts 0-length and excessively long addresses	Data Validation	**Informational**
4	Invalid ConsensusStateWithHeight.ConsensusState struct tag could cause incorrect or unexpected serialization results	Data Validation	**Undetermined**
5	Deserializing untrusted cosmos transactions with the legacy amino codec can crash the program	Data Validation	**High**
6	Incorrectly formatted error string in OnChanOpenTry function	Auditing and Logging	**Low**
7	If the host fails to execute a single message, none of the messages sent by the controller in a single packet are committed	Timing	**Undetermined**
8	logged targetClient object is always nil when error is returned	Auditing and Logging	**Informational**
9	The validateEnabled implementations check only the passed-in variable type and not its value, as some usages suggest it should	Auditing and Logging	**Undetermined**
10	The host chain returns a single error instead of all aggregated errors for executed messages	Auditing and Logging	**Low**

Detailed Findings

1. Outdated and vulnerable dependencies	
Severity: **Informational**	Difficulty: **N/A**
Type: Patching	Finding ID: TOB-IBICA-1
Target: `ibc-go/go.mod`	

Description

The `ibc-go` repository contains outdated and vulnerable dependencies that can be detected (e.g., with GitHub's Dependabot). We list these problematic packages in the table below. This finding is of informational severity, as the listed bugs do not seem to impact the current codebase.

Package	Vulnerability and Severity	Bug
`github.com/gin-gonic/gin < 1.7.0`	CVE-2020-28483 (High)	*"When gin is exposed directly to the internet, a client's IP can be spoofed by setting the X-Forwarded-For header." (source)* The problematic `.ClientIp()` function is not called directly by the `ibc-go` codebase.
`github.com/opencontainers/image-spec < 1.0.2`	CVE-2021-41190 and GHSA-77vh-xpmg-72qh (Low)	*"In the OCI Image Specification version 1.0.1 and prior, manifest and index documents are not self-describing and documents with a single digest could be interpreted as either a manifest or an index."*
`github.com/opencontainers/runc < 1.0.3`	CVE-2021-43784 (Moderate)	A malicious container configuration could bypass the namespace restrictions in an unintended way.

Exploit Scenario

An attacker fingerprints the service, identifies an out-of-date package with a known vulnerability, and uses a public exploit against the service.

Recommendations
Short term, update the Golang dependencies in the `ibc-go` repository, as this repository depends on outdated packages with known vulnerabilities.

2. Revision and client identifier regex formats accept newlines, which may be unintended behavior

Severity: **Undetermined**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-IBCICA-2
Target: `ibc-go/modules/core/02-client/types/{height,keys}.go`	

Description

The `IsRevisionFormat` and `IsClientIDFormat` regular expressions (regexes) implemented in the `02-client` module use the "`[^-]`" regex element to ensure that two components are delimited by only a single dash in the respective `{chainID}-{revision}` and `{client-type}-{N}` formats. However, this regex element also accepts newline characters, which could cause unwanted “revision” and “client identifier” formats to be accepted.

```
// IsRevisionFormat checks if a chainID is in the format required for parsing revisions
// The chainID must be in the form: `{chainID}-{revision}
// 24-host may enforce stricter checks on chainID
var IsRevisionFormat = regexp.MustCompile(`^.*[^-]-{1}[1-9][0-9]*$`).MatchString
```

Figure 2.1: `ibc-go/modules/core/02-client/types/height.go#L17-L20`

```
// IsClientIDFormat checks if a clientID is in the format required on the SDK for
// parsing client identifiers. The client identifier must be in the form: `{client-type}-{N}
var IsClientIDFormat = regexp.MustCompile(`^.*[^-]-[0-9]{1,20}$`).MatchString
```

Figure 2.2: `ibc-go/modules/core/02-client/types/keys.go#L34-L36`

Recommendations

Short term, fix the `IsRevisionFormat` and `IsClientIDFormat` regexes by changing the "`[^-]`" element to "`[^\n-]`" so that newline characters are not accepted before the "`-`" delimiter. Alternatively, if newline characters should be accepted, document this fact in the code comment.

Long term, extend the `TestParseClientIdentifier` and `TestParseChainID` test cases to check for client and chain IDs that contain newline characters before the "`-`" delimiter.

3. The IsValidAddr regex accepts 0-length and excessively long addresses

Severity: **Informational**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-IBCICA-3
Target: `ibc-go/modules/apps/27-interchain-accounts/types/version.go`	

Description

The `IsValidAddr` regex (figure 3.1) accepts `0`-length addresses and addresses longer than the `DefaultMaxAddrLength`, which are not valid. The `ValidateAccountAddress` function mitigates this issue by erroring out if the address string length is `0` or if it is longer than the `DefaultMaxAddrLength`; however, because the regex is a public variable, other developers could use this regex directly instead of calling the `ValidateAccountAddress` function, introducing data validation bugs into the software.

```
// IsValidAddr defines a regular expression to check if the provided string consists of
// strictly alphanumeric characters
var IsValidAddr = regexp.MustCompile("^[a-zA-Z0-9]*$").MatchString
```

Figure 3.1:
`ibc-go/modules/apps/27-interchain-accounts/types/version.go#L14-L16`

```
// ValidateAccountAddress performs basic validation of interchain account addresses,
enforcing constraints
// on address length and character set
func ValidateAccountAddress(addr string) error {
	if !IsValidAddr(addr) || len(addr) == 0 || len(addr) > DefaultMaxAddrLength {
		return sdkerrors.Wrapf(
			ErrInvalidAccountAddress,
			"address must contain strictly alphanumeric characters, not exceeding
%d characters in length",
			DefaultMaxAddrLength,
		)
	}

	return nil
}
```

Figure 3.2:
`ibc-go/modules/apps/27-interchain-accounts/types/version.go#L55-L67`

Recommendations

Short term, change the `IsValidAddr` regex to use a "+" quantifier instead of "*" so that it will match 1 or more characters, rather than `0` or more characters, from the `[a-zA-Z0-9]` range. Additionally, consider either modifying the regex so that it takes `DefaultMaxAddrLength` into account or lowercasing the regex's name to make it a private variable. This will help prevent bugs in the future if someone uses the regex directly instead of through the `ValidateAccountAddress` function.

4. Invalid ConsensusStateWithHeight.ConsensusState struct tag could cause incorrect or unexpected serialization results

Severity: **Undetermined**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-IBCICA-4
Target: `ibc-go/modules/core/02-client/types/client.pb.go:90:2`	

Description

Upon running `go vet`, we discovered that the protobuf-generated structure `ConsensusStateWithHeight` has an invalid struct tag for its `ConsensusState` field. The tag is missing a colon (`:`) character immediately after the `yaml` key. This invalid struct tag may lead to incorrect or unexpected serialization of the `ConsensusState` field in all three formats defined in the tag (`protobuf`, `json`, and `yaml`).

This problem is caused by the same typo (the missing colon character before `yaml` key) in the protobuf source file, shown in figure 4.2. Appendix B provides an example of the impact of this issue.

```
 type ConsensusStateWithHeight struct {
   // consensus state height
   Height Height `protobuf:"bytes,1,opt,name=height,proto3" json:"height"`
   // consensus state
   ConsensusState *types.Any
`protobuf:"bytes,2,opt,name=consensus_state,json=consensusState,proto3"
json:"consensus_state,omitempty" yaml"consensus_state"`
}
```

Figure 4.1: `modules/core/02-client/types/client.pb.go#L86-L91`

```
message ConsensusStateWithHeight {
 // consensus state height
 Height height = 1 [(gogoproto.nullable) = false];
 // consensus state
 google.protobuf.Any consensus_state = 2 [(gogoproto.moretags) = "yaml\"consensus_state\""];
}
```

Figure 4.2: `ibc-go/proto/ibc/core/client/v1/client.proto#L20-L27`

Recommendations

Short term, fix the invalid Go struct tag in the `ibc-go/proto/ibc/core/client/v1/client.proto` file by adding the missing colon

(`:`) character immediately after the `yaml` key in the `ConsensusState` field of the `ConsensusStateWithHeight` message format.

Long term, use the `go vet` tool in the CI/CD pipeline to detect issues with invalid struct tags during the development process.

5. Deserializing untrusted cosmos transactions with the legacy amino codec can crash the program

Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-IBCICA-5
Target: `ibc-go/modules/apps/27-interchain-accounts/types/codec.go`	

Description

The `DeserializeCosmosTx` function can panic and crash the program if it is used with the legacy amino codec and a specially crafted input.

We found this bug by fuzzing the `DeserializeCosmosTx` function with the legacy amino and protobuf codecs. The fuzzing harness we developed and instructions on running it are provided in Appendix C.

This problem can be reproduced with the test in figure 5.1, which can be added directly to the `ibc-go/modules/apps/27-interchain-accounts/types/codec_test.go` file. Figure 5.2 shows a stack trace from running this test.

```
func (suite *TypesTestSuite) TestDeserializeCosmosTxNotCrashes() {
	cdc := codec.NewLegacyAmino()
	marshaler := codec.NewAminoCodec(cdc)
	types.DeserializeCosmosTx(marshaler, []byte{0x10, 0})
}
```

Figure 5.1: A test that reproduces the `DeserializeCosmosTx` function crash

```
=== RUN   TestTypesTestSuite/TestDeserializeCosmosTxNotCrashes
    suite.go:63: test panicked: runtime error: invalid memory address or nil pointer
dereference
        goroutine 51 [running]:
        runtime/debug.Stack()
              /usr/local/go/src/runtime/debug/stack.go:24 +0x65
        github.com/stretchr/testify/suite.failOnPanic(0xc00057c1a0)
              /home/dc/go/pkg/mod/github.com/stretchr/testify@v1.7.0/suite/suite.go:63
+0x3e
        panic({0x15a9c80, 0x2811190})
              /usr/local/go/src/runtime/panic.go:1038 +0x215
        github.com/cosmos/cosmos-sdk/codec/types.AminoUnpacker.UnpackAny({0xc000d8f4e0},
0x40bb33, {0x14fc8a0, 0xc001121e40})
// (...)
    --- FAIL: TestTypesTestSuite/TestDeserializeCosmosTxNotCrashes (0.06s)
```

Figure 5.2: A stack trace from the crash from running the test shown in figure 5.1

Exploit Scenario
Alice implements a blockchain that uses IBC's interchain accounts module and serializes cosmos transactions using the legacy amino codec. Eve, who knows that Alice has done this, creates a fake controller chain, connects it to Alice's chain via IBC relayers, and triggers a denial-of-service attack on Alice's blockchain nodes by sending a malicious cosmos transaction that causes the nodes to crash.

Recommendations
Short term, investigate and fix the issue that causes the `DeserializeCosmosTx` function to panic and crash the program when it is used with the legacy amino codec and called with untrusted input. Alternatively, if the function is not intended to be used with untrusted input, add a documentation string that explains this intention and the risks associated with passing untrusted input to this function.

Long term, run the fuzzing harness in Appendix C for a longer period of time to ensure that the codebase does not contain similar cases. Additionally, consider implementing fuzzing for other parts of the system.

6. Incorrectly formatted error string in OnChanOpenTry function	
Severity: **Low**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-IBCICA-6
Target: `ibc-go/…/27-interchain-accounts/host/keeper/handshake.go`	

Description
The `OnChanOpenTry` function in the host validates the passed `counterpartyVersion`. If the value is invalid, the function returns an incorrectly formatted error string. This string indicates that the function checked the value of the `version` string argument rather than the value of the `counterpartyVersion` string argument. This error may be confusing for users.

```
func (k Keeper) OnChanOpenTry(/* (...) */ version, counterpartyVersion string) error {
    // (...)
    if counterpartyVersion != icatypes.VersionPrefix {
        return sdkerrors.Wrapf(icatypes.ErrInvalidVersion, "expected %s, got %s",
icatypes.VersionPrefix, version)
    }
```

Figure 6.1: *`ibc-go/modules/apps/27-interchain-accounts/host/keeper/handshake.go#L54-L56`*

Exploit Scenario
Eve wants to integrate with the IBC interchain accounts module. She sends the `ChanOpenTry` message from a controller to the host chain with an incorrect `counterpartyVersion` string. She then loses a lot of time investigating why she received an error stating that the provided `version` string, rather than the `counterpartyVersion` string, is invalid.

Recommendations
Short term, in the `OnChanOpenTry` function in the `host/keeper/handshake.go` file, fix the formatted error message that returns when a user passes an incorrect `counterpartyVersion`. The error should indicate that the `counterpartyVersion`, instead of the `version`, is incorrect. This will prevent confusion in users who incorrectly use the interchain accounts API.

Long term, extend the test suite to check for error cases and assert the received error messages.

7. If the host fails to execute a single message, none of the messages sent by the controller in a single packet are committed

Severity: **Undetermined**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-IBCICA-7
Target: `ibc-go/modules/core/keeper/msg_server.go`	

Description

The `executeTx` function, which validates and executes the received messages on the host chain, errors out if the validation or execution of any message fails; if the function errors out for this reason, it does not commit any of the state changes. While this behavior may be desirable, it could allow an attacker to cancel the execution of all messages sent to the host by front-running the execution of the messages to trigger a failure.

This finding is of undetermined severity, as we have not fully confirmed whether the described exploit scenario is possible.

```
func (k Keeper) executeTx(ctx sdk.Context, sourcePort, destPort, destChannel string, msgs
[]sdk.Msg) error {
	// (...)
	cacheCtx, writeCache := ctx.CacheContext()
	for _, msg := range msgs {
		if err := msg.ValidateBasic(); err != nil {
			return err
		}
		if _, err := k.executeMsg(cacheCtx, msg); err != nil {
			return err
		}
	}
	writeCache()
	return nil
}
```

Figure 7.1: `ibc-go/modules/apps/27-interchain-accounts/host/keeper/relay.go#L36-L57`

Exploit Scenario

1. Alice sends two messages, A and B, which execute certain transactions on behalf of an interchain account that she owns.

2. The A and B messages do not interfere with each other. However, A depends on the chain state and may fail if it is front-run with a specific transaction; B is expected to succeed, as it depends only on a state that is controlled only by Alice.
3. Eve observes the messages initiated by Alice on the controller chain and wants the B transaction to fail. Eve cannot influence the B transaction directly; however, she can send a transaction that will change an on-chain state that the A transaction depends on, causing the A transaction to fail.
4. Eve front-runs the A transaction with her own state-changing transaction.
5. The host chain executes Alice's messages, and due to Eve's transaction, the execution stops after the failure of Alice's A transaction.
6. Since the host chain does not proceed with executing the remaining messages, Alice's B transaction is not executed at all.

Note that the order in which the A and B messages are sent and executed does not matter here, as the `executeTx` function commits the modified chain state only if it successfully executes all messages.

Recommendations
Short term, investigate the system's behavior when it fails to execute a single message. Modify the message execution logic so that the system proceeds with executing messages after a failure occurs, or so that the system's handling of messages is configurable by the sender. Alternatively, leave the message execution logic "as is," but document this issue. Inform blockchain developers of the risks of sending multiple interchain account messages at once rather than one at a time.

Long term, test the interchain accounts execution logic against front-running scenarios.

8. logged targetClient object is always nil when error is returned

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-IBCICA-8
Target: `ibc-go/modules/core/keeper/msg_server.go`	

Description

The `Keeper.ConnectionOpenTry` function calls `clienttypes.UnpackClientState` to obtain a target client (figure 8.1). When `clienttypes.UnpackClientState` returns an error, the received `targetClient` object is indicated in the error logs. However, `clienttypes.UnpackClientState`'s errors always indicate `nil` for the first returned value (figure 8.2). As a result, the returned error will always indicate a `<nil>` target client. This reduces the utility of the error. Moreover, based on the error message format (and the `%v` format specifier), it appears that the developers may have intended the function to log the `msg.ClientState` instead of the `targetClient`.

```go
func (k Keeper) ConnectionOpenTry(goCtx context.Context, msg
*connectiontypes.MsgConnectionOpenTry) (*connectiontypes.MsgConnectionOpenTryResponse,
error) {
	ctx := sdk.UnwrapSDKContext(goCtx)

	targetClient, err := clienttypes.UnpackClientState(msg.ClientState)
	if err != nil {
		return nil, sdkerrors.Wrapf(err, "client in msg is not exported.ClientState.
invalid client: %v.", targetClient)
	}
```

Figure 8.1: `ibc-go/modules/core/keeper/msg_server.go#L159-L165`

```go
func UnpackClientState(any *codectypes.Any) (exported.ClientState, error) {
  if any == nil {
    return nil, sdkerrors.Wrap(sdkerrors.ErrUnpackAny, "protobuf Any message cannot be nil")
  }

  clientState, ok := any.GetCachedValue().(exported.ClientState)
  if !ok {
    return nil, sdkerrors.Wrapf(sdkerrors.ErrUnpackAny, "cannot unpack Any into ClientState
%T", any)
  }

  return clientState, nil
}
```

Figure 8.2: `ibc-go/modules/core/02-client/types/codec.go#L70-L83`

Recommendations

Short term, correct the `Keeper.ConnectionOpenTry` function so that, on error, it returns a message with the intended object.

Long term, include Semgrep and the rule in Appendix D to the CI/CD process to catch these issues when the code is pushed to the IBC-Go repository.

9. The validateEnabled implementations check only the passed-in variable type and not its value, as some usages suggest it should

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Auditing and Logging	Finding ID: TOB-IBCICA-9
Target: Various	

Description

There are three instances of the `validateEnabled` function defined in separate `types` packages. Despite this function's name and usage throughout the codebase, the function checks only whether the passed-in parameter, a generic interface type, is a boolean. Given its implementation, the function returns a `nil` error as long as its only argument is a boolean, regardless of whether that boolean is `true` or `false`.

The duplicated `validateEnabled` implementations are in the following files:

- `ibc-go/modules/apps/transfer/types/params.go#L58-L65`
- `ibc-go/modules/apps/27-interchain-accounts/controller/types/params.go#L52-L59`
- `ibc-go/modules/apps/27-interchain-accounts/host/types/params.go#L61-L68`

In certain locations, this function is called with a boolean variable, suggesting either that the function should check the value of the variable (not only its type) or that the calls are unnecessary. Figure 9.2 shows an example of a problematic call to `validateEnabled`, and figure 9.3 shows all the calls to `validateEnabled` in the codebase.

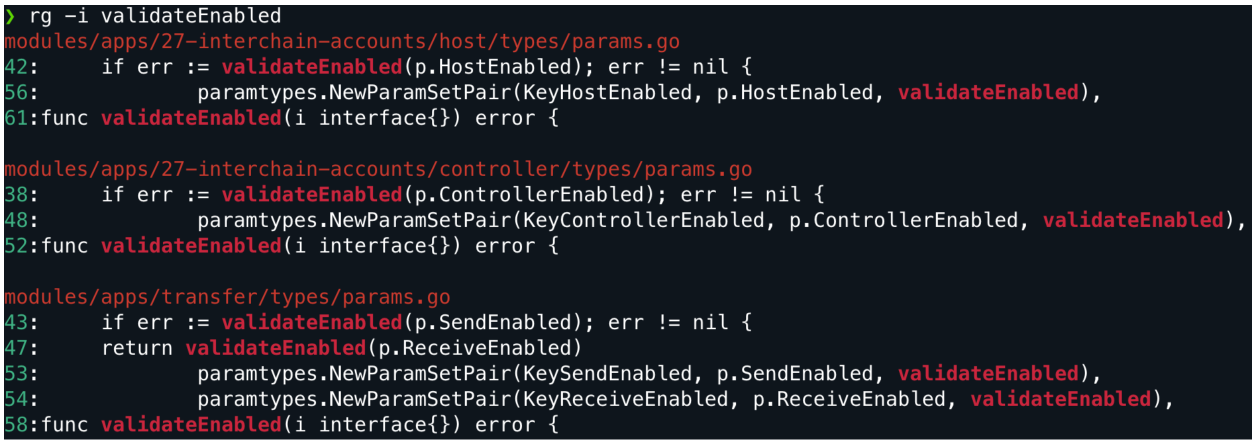
```
func validateEnabled(i interface{}) error {
	_, ok := i.(bool)
	if !ok {
		return fmt.Errorf("invalid parameter type: %T", i)
	}

	return nil
}
```

Figure 9.1: `ibc-go/modules/apps/transfer/types/params.go#L58-L65`

```
func (p Params) Validate() error {
        // Note: the p.ControllerEnabled is bool and this check never returns an error
        if err := validateEnabled(p.ControllerEnabled); err != nil {
                return err
        }
        return nil
}
```

Figure 9.2:
`ibc-go/modules/apps/27-interchain-accounts/controller/types/params.go#L36-L43`



```
❯ rg -i validateEnabled
modules/apps/27-interchain-accounts/host/types/params.go
42:     if err := validateEnabled(p.HostEnabled); err != nil {
56:             paramtypes.NewParamSetPair(KeyHostEnabled, p.HostEnabled, validateEnabled),
61:func validateEnabled(i interface{}) error {

modules/apps/27-interchain-accounts/controller/types/params.go
38:     if err := validateEnabled(p.ControllerEnabled); err != nil {
48:             paramtypes.NewParamSetPair(KeyControllerEnabled, p.ControllerEnabled, validateEnabled),
52:func validateEnabled(i interface{}) error {

modules/apps/transfer/types/params.go
43:     if err := validateEnabled(p.SendEnabled); err != nil {
47:     return validateEnabled(p.ReceiveEnabled)
53:             paramtypes.NewParamSetPair(KeySendEnabled, p.SendEnabled, validateEnabled),
54:             paramtypes.NewParamSetPair(KeyReceiveEnabled, p.ReceiveEnabled, validateEnabled),
58:func validateEnabled(i interface{}) error {
```

Figure 9.3: The locations in which `validateEnabled` is called

Exploit Scenario
A developer uses the `validateEnabled` function, believing that it checks whether a variable such as `p.ControllerEnabled` is set to `true`. However, because the function simply checks whether the passed-in value is a boolean, the resulting logic is flawed, leading to unexpected results and invalid logic.

Recommendations
Short term, correct the implementation of the `validateEnabled` function. Furthermore, avoid repeating the same logic throughout the codebase; rather, define the same function in a single package so that it can be used anywhere that it is needed in the codebase.

10. The host chain returns a single error instead of all aggregated errors for executed messages

Severity: **Low**	Difficulty: **Undetermined**
Type: Auditing and Logging	Finding ID: TOB-IBCICA-10
Target: Various	

Description

When the host chain validates and executes the received messages in the `AuthenticateTx` and `executeTx` functions (figure 10.1), it returns early if it encounters an error for any message. As a result, the user of this logic will not get all the errors resulting from the processing of sent messages.

```
func (k Keeper) AuthenticateTx(ctx sdk.Context, msgs []sdk.Msg, portID string) error {
	interchainAccountAddr, found := k.GetInterchainAccountAddress(ctx, portID)
	if !found {
		return sdkerrors.Wrapf(icatypes.ErrInterchainAccountNotFound, "failed to
retrieve interchain account on port %s", portID)
	}

	allowMsgs := k.GetAllowMessages(ctx)
	for _, msg := range msgs {
		if !types.ContainsMsgType(allowMsgs, msg) {
			return sdkerrors.Wrapf(sdkerrors.ErrUnauthorized, "message type not
allowed: %s", sdk.MsgTypeURL(msg))
		}

		for _, signer := range msg.GetSigners() {
			if interchainAccountAddr != signer.String() {
				return sdkerrors.Wrapf(sdkerrors.ErrUnauthorized, "unexpected
signer address: expected %s, got %s", interchainAccountAddr, signer.String())
			}
		}
	}

	return nil
}

func (k Keeper) executeTx(ctx sdk.Context, sourcePort, destPort, destChannel string, msgs
[]sdk.Msg) error {
	if err := k.AuthenticateTx(ctx, msgs, sourcePort); err != nil {
		return err
	}
	// (...)
	cacheCtx, writeCache := ctx.CacheContext()
	for _, msg := range msgs {
		if err := msg.ValidateBasic(); err != nil {
			return err
		}

		if _, err := k.executeMsg(cacheCtx, msg); err != nil {
			return err
		}
	}

	writeCache()

	return nil
}
```

Figure 10.1: `ibc-go/modules/apps/transfer/types/params.go#L58-L65`

Exploit Scenario
Alice sends two messages from the controller chain to the host chain. Both messages are invalid, but the host chain returns only the error for the first message. Alice corrects her error and resends the two messages with the corrected first message. The host chain returns the second error. Alice corrects her second message and resends the two corrected messages. The host chain executes both messages. As a result of having to resend the messages multiple times, Alice has wasted funds.

Recommendations
Short term, consider calculating and aggregating all message validation errors in the `AuthenticateTx` and `executeTx` functions so that the users of this logic are aware of all issues with the sent messages.

Summary of Recommendations

Trail of Bits recommends that Interchain Berlin address the findings detailed in this report and take the following additional steps prior to deployment:

- Expand the unit testing suite, and focus on testing functions with unexpected input.
- Consider incorporating property-based testing tools such as Gopter into the testing strategy. Property-based testing can aid in constructing tests that verify the desired properties of the IBC protocol.
- Fuzz complex parsing, serialization, and deserialization logic to catch issues such as TOB-IBCICA-5, which could cause application crashes.
- If any of the code paths associated with the findings in this report are refactored, perform an additional internal review of the invariants.
- Regularly run static analysis tools such as `go vet` and Semgrep to uncover common correctness issues, such as those described in TOB-IBCICA-4 and TOB-IBCICA-9, and incorporate these tools into the CI/CD pipeline.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories

Category	Description
Access Controls	Insufficient authorization of users or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	Breach of the confidentiality or integrity of data
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	System failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions, locking, or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defense in depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the client has indicated is important.
Medium	Individual users' information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

B. Example of Marshaling Malformed Struct Tags

This appendix provides an example of the problem described in finding TOB-IBCICA-4, in which the structure tags were malformed.

Figure B.1 shows a code snippet with structures containing valid and invalid tags. The structures are marshaled to JSON and printed as strings. In figure B.2, we can see the output of this example, which shows that the malformed tags were not used.

```go
package main

import (
    "fmt"
    "encoding/json"
)

type ProperTag struct {
        SomeKey string `json:"some_key"`            // Valid tags
}

type MissingQuotationInvalidTag struct {
        SomeKey string `json:"some_key`             // Invalid: missing end " character
}

type RedundantSpaceInvalidTag struct {
        SomeKey string `json: "some_key"`           // Invalid: redundant space after 'json:'
}

type RedundantSpaceOmitemptyInvalidTag struct {
        SomeKey string `json:"some_key, omitempty"` // Invalid: redundant space after ','
}

type ProperOmitemptyTag struct {
        SomeKey string `json:"some_key,omitempty"`  // Valid tag
}

func Print(jsondata []byte, _ error) {
    fmt.Println(string(jsondata))
}

func main() {
    Print(json.Marshal(ProperTag { "a" }))
    Print(json.Marshal(MissingQuotationInvalidTag { "a" }))
    Print(json.Marshal(RedundantSpaceInvalidTag { "a" }))
    Print(json.Marshal(RedundantSpaceInvalidTag { "" }))
    Print(json.Marshal(ProperOmitemptyTag { "" }))
}
```

Figure B.1: Example of malformed struct tags

```
$ go run main.go
{"some_key":"a"}                      // Tag: `json:"some_key"`
{"SomeKey":"a"}                       // Tag: `json:"some_key`
```

```
{"SomeKey":"a"}                        // Tag: `json: "some_key"`
{"SomeKey":""}                         // Tag: `json:"some_key, omitempty"`
{}                                     // Tag: `json:"some_key,omitempty"`
```

Figure B.2: The output of the code in figure B.1 (the yellow highlight is our comment)

C. Fuzzing the DeserializeCosmosTx Function

During the audit, Trail of Bits used fuzzing, an automated testing technique in which code paths are executed with random data to find bugs resulting from the incorrect handling of unexpected data. We used `dvyukov/go-fuzz`, an in-process coverage-guided fuzzer for Golang, to develop fuzzing harnesses for the `DeserializeCosmosTx` function. These harnesses helped us to find the issue in TOB-IBCICA-5, in which untrusted input passed to the `DeserializeComsosTx` function can crash the program if the legacy amino codec is used.

We ran the harnesses for a limited period of time. We recommend running them further, such as until the fuzzer does not find input generating new coverage for a few hours or longer. In such a case, we recommend investigating the coverage of all corpus input that the fuzzer generated by creating a small program that executes the fuzzed function with a given payload and instrumenting it for code coverage. This could help to find code paths that were not executed and to manually craft or modify the corpus to achieve higher coverage and find new bugs.

In the sections below, we introduce the `go-fuzz` fuzzer that we used and show the fuzzing harnesses that we developed.

Fuzzing with dvyukov/go-fuzz 101

The `dvyukov/go-fuzz` package provides an AFL-like mutational fuzzing interface, in which testing harnesses can be built entirely in Go. This framework is typically used when a library implemented in Go parses, interprets, or otherwise interacts with blobs of data. An example of such a use case can be seen in figure C.1, in which a harness for the Go standard library's image-processing library is defined.

```go
package png

import (
	"bytes"
	"image/png"
)

func Fuzz(data []byte) int {
	png.Decode(bytes.NewReader(data))
	return 0
}
```

Figure C.1: An example test harness for the `png.Decode` function, shown in the official readme

In this example, the function `Fuzz` accepts an array of bytes for `data`, which is then converted into a `Reader` for the `png.Decode` function to read from. When `Fuzz` is compiled and invoked, it is executed repeatedly, using `data` as the input generated for each test case execution.

Optimizing `go-fuzz`'s generation of test case input requires an understanding of return values. Typically, a panic indicates a crash with a given test case input. However, when no crash occurs, but instead no errors are raised, or errors are raised gracefully, return values can be used to help guide `go-fuzz` to mutate input appropriately.

- A return value of 1 indicates the input generator should increase the priority of a given input during subsequent fuzzing.
- A return value of `-1` indicates the input generator should never be added to the corpus, despite added coverage.
- In all other cases, the function should return `0`.

Installing and Running dvyukov/go-fuzz

To build and run this example, you must have Go and the `go-fuzz` package installed. You can then navigate to the directory in which the test harness in figure C.1 is stored and execute `go-fuzz-build` (figure C.2). Assuming the harness builds correctly, it will produce a Zip file for use with the `go-fuzz` executor. To start the fuzzing harness, you can execute `go-fuzz` in the same directory as the Zip file produced by `go-fuzz-build` (figure C.3). This will create three directories, if they do not already exist.

```
user@host:~/Desktop/png_fuzz$ ls
png_harness.go
user@host:~/Desktop/png_fuzz$ go-fuzz-build
user@host:~/Desktop/png_fuzz$ ls
png-fuzz.zip  png_harness.go
```

Figure C.2: The generated `png-fuzz.zip` package used by `go-fuzz`

```
user@host:~/Desktop/png_fuzz$ go-fuzz
2019/09/14 16:00:37 workers: 2, corpus: 30 (0s ago), crashers: 0, restarts: 1/0, execs: 0
(0/sec), cover: 0, uptime: 3s
2019/09/14 16:00:40 workers: 2, corpus: 31 (2s ago), crashers: 0, restarts: 1/0, execs: 0
(0/sec), cover: 205, uptime: 6s
2019/09/14 16:00:43 workers: 2, corpus: 31 (5s ago), crashers: 0, restarts: 1/6092, execs:
48742 (5415/sec), cover: 205, uptime: 9s
2019/09/14 16:00:46 workers: 2, corpus: 31 (8s ago), crashers: 0, restarts: 1/7829, execs:
101779 (8481/sec), cover: 205, uptime: 12s
```

```
2019/09/14 16:00:49 workers: 2, corpus: 31 (11s ago), crashers: 0, restarts: 1/8147, execs:
146656 (9777/sec), cover: 205, uptime: 15s
2019/09/14 16:00:52 workers: 2, corpus: 31 (14s ago), crashers: 0, restarts: 1/8851, execs:
203582 (11310/sec), cover: 205, uptime: 18s
2019/09/14 16:00:55 workers: 2, corpus: 31 (17s ago), crashers: 0, restarts: 1/8950, execs:
259563 (12360/sec), cover: 205, uptime: 21s
^C2019/09/14 16:00:56 shutting down...
```

Figure C.3: The CLI output of running `go-fuzz` with the `png-fuzz.zip` package

The created directories contain suppressions, crashers, and a corpus, respectively (figure C.4). The suppressions are used to prevent the same message values from being collected every time the fuzzer runs, polluting your crasher samples. The crashers are the program's crashdumps—the STDOUT and STDERR of the program when the test case input causes an error. Finally, the corpus directory stores the test case inputs used throughout the test harness's execution. This directory will collect mutated versions of each input as necessary.

```
user@host:~/Desktop/png_fuzz$ ls -R
.:
corpus  crashers  png-fuzz.zip  png_harness.go  suppressions

./corpus:
21339f0e4b8b5a8e0cb5471f1f91907d1917be50-6
215d99d0c7acdec5ad4c5aa8bec96a171b9ffae0-8
22f545ac6b50163ce39bac49094c3f64e0858403-11
// (...)

./crashers:

./suppressions:
```

Figure C.4: The directory and file output produced by `go-fuzz`

While running the harness on a single machine typically produces good results, `go-fuzz` also supports a clustered mode, allowing test harness execution to scale horizontally across an arbitrary number of worker nodes. More information on this functionality can be found within the repository's readme.

Fuzzing Harnesses for the DeserializeCosmosTx Function

Figure C.5 shows the fuzzing harnesses developed for the `DeserializeCosmosTx` function that helped us to find the bug described in TOB-IBCICA-5. These harnesses can be run with the following steps:

1. Install the `go-fuzz` fuzzer, as described in the previous section.
2. Create a `ibc-go/modules/apps/27-interchain-accounts/types/fuzz.go` file with the content in figure C.5.

3. In the same directory as the `fuzz.go` file, execute the `go-fuzz-build` command to build the fuzzing Zip archive.
4. Run the harnesses by using the commands in figure C.6, which also shows the final log line from our runs of the harnesses. You could run these harnesses on a dedicated server in a `tmux` or `screen` session to be able to detach from the terminal and let the fuzzer run in background.

```
package types

import (
	"github.com/cosmos/cosmos-sdk/codec"
	"github.com/cosmos/cosmos-sdk/codec/types"
)

func FuzzDeserializeCosmosTxAmino(data []byte) int {
	cdc := codec.NewLegacyAmino()
	marshaler := codec.NewAminoCodec(cdc)
	DeserializeCosmosTx(marshaler, data)
	return 0
}

func FuzzDeserializeCosmosTxProto(data []byte) int {
	interfaceRegistry := types.NewInterfaceRegistry()
	marshaler := codec.NewProtoCodec(interfaceRegistry)
	DeserializeCosmosTx(marshaler, data)
	return 0
}
```

Figure C.5: The developed fuzzing harnesses for the `DeserializeCosmosTx` function

```
$ ~/go/bin/go-fuzz -bin types-fuzz.zip -workdir FuzzDeserializeCosmosTxProto -procs 2 -func
FuzzDeserializeCosmosTxProto
// (...)
2021/12/10 07:07:32 workers: 2, corpus: 178 (3m50s ago), crashers: 0, restarts: 1/9998,
execs: 83234512 (4829/sec), cover: 396, uptime: 4h47m

$ ~/go/bin/go-fuzz -bin types-fuzz.zip -workdir FuzzDeserializeCosmosTxAmino -procs 2 -func
FuzzDeserializeCosmosTxAmino
// (...)
2021/12/10 07:07:43 workers: 2, corpus: 99 (1h26m ago), crashers: 1, restarts: 1/680, execs:
25583828 (1482/sec), cover: 1091, uptime: 4h47m
```

Figure C.6: The log from our runs of the fuzzing harnesses (note that we ran them with only two CPUs and by setting their `workdirs`, so all metadata like crasher files will appear in those directories)

Getting Initial Corpus Files for the DeserializeCosmosTx Function

To help the fuzzer discover more paths into the `DeserializeCosmosTx` function, we also made a small and temporary modification to the function's code, shown in figure C.7, so that it saves all its inputs to a temporary directory. We then ran all the unit tests in the codebase and got a few input files that we used as an initial corpus for the fuzzing harnesses in figure C.5. (The modifications were removed before compiling the fuzzing harnesses).

```go
func DeserializeCosmosTx(cdc codec.BinaryCodec, data []byte) ([]sdk.Msg, error) {
	f, err := os.CreateTemp("/tmp/cosmostx/", fmt.Sprintf("%s.*", reflect.TypeOf(cdc)))
	if err != nil { panic("err: CreateTemp") }
	_, err = f.Write(data)
	if err != nil { panic("err: f.Write") }
	if f.Close() != nil { panic("err: f.Close") }
```

Figure C.7: Modifications made to the `DeserializeCosmosTx` function to save initial corpus files

D. Semgrep Rules

This appendix lists Semgrep rules that we used to discover the issues described in this report.

Invalid Usage of Modified Variables

The following rule was used to discover TOB-IBCICA-8.

```yaml
rules:
  - id: invalid-usage-of-modified-variable
    patterns:
      - pattern-either:
        - pattern: |
            $X, err = ...
            if err != nil {
              <... $FOO(..., $X, ...) ...>
            }
    message: $X is likely modified and later used on error
    languages: [go]
    severity: WARNING
```

Figure D.1: Semgrep rule to find potentially invalid usage of modified variables

E. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- In the `proto/ibc/core/channel/v1/tx.proto#L59-L61` file, the `MsgChannelOpenTry` message has an incorrect comment. This comment was most likely copied from the `MsgChannelOpenInit` message:

```
// MsgChannelOpenInit defines a msg sent by a Relayer to try to open a channel
// on Chain B.
message MsgChannelOpenTry {
```

  We recommend fixing this comment so that it does not confuse developers who read it.

- `modules/light-clients/07-tendermint/types/misbehaviour_handle.go` contains a `clienttypes.IsRevisionFormat(chainID)` check that guards a `clienttypes.SetRevisionNumber(...)` call, which performs the same check:

```
if clienttypes.IsRevisionFormat(chainID) {
        chainID, _ = clienttypes.SetRevisionNumber(chainID,
header.GetHeight().GetRevisionNumber())
}
```

ibc-go/modules/light-clients/07-tendermint/types/misbehaviour_handle.go#L132-L134

```
func SetRevisionNumber(chainID string, revision uint64) (string, error) {
        if !IsRevisionFormat(chainID) {
                return "", sdkerrors.Wrapf(
                        sdkerrors.ErrInvalidChainID, "chainID is not in revision
format: %s", chainID,
                )
        }
        // (...)
```

ibc-go/modules/core/02-client/types/height.go#L152-L157

  We recommend refactoring the `misbehaviour_handle.go#L132-L134` and `update.go#L215-L217` files. The logic should call the `SetRevisionNumber` function unconditionally and handle the potential error result instead of only performing the `IsRevisionFormat` check (which is done in the call to `SetRevisionNumber`).