ZetaChain

Security Assessment

October 3, 2024

Prepared for:
Lucas Bertrand
ZetaChain

Prepared by: **Simone Monica and Troy Sargent**

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 100+ 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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

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 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 its partners. As a result, 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.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Simone Monica, Consultant
simone.monica@trailofbits.com

Troy Sargent, Consultant
troy.sargent@trailofbits.com

Project Timeline

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

Date	Event
July 25, 2024	Pre-project kickoff call
August 5, 2024	Status update meeting #1
August 12, 2024	Delivery of report draft
August 12, 2024	Report readout meeting
October 3, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

The ZetaChain core team engaged Trail of Bits to review the security of changes applied to its ZetaChain Cosmos node codebase. ZetaChain is a Cosmos-based network that facilitates the bridging and exchange of tokens between itself and EVM/non-EVM chains. ZetaChain's funds on other chains are administered using a TSS-based threshold key whose parts are held by a class of permissioned Cosmos validators called "observers."

A team of two consultants conducted the review from July 29, 2024 to August 9, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on possible issues introduced by the new changes, such as message authorization bypass or double-spend attacks. With full access to source code and documentation, we performed static and dynamic testing of the ZetaChain node, using automated and manual processes. Our review focused on the Cosmos SDK version upgrade (PR#2100), authorization refactoring (PR#2289 and PR#2313), and CCTX processing refactoring (PR#2317 and PR#2340).

Observations and Impact

Given that ZetaChain is a complex system, implementing changes that make the code easier to develop—in particular, CCTX inbound and outbound validation logic refactoring—is an important investment.

Although we noted refactoring and additional documentation that help to simplify the validation logic, the documentation flow and variables included in the processing of a cross-chain transaction are still subpar (TOB-ZETA-12).

Our review found multiple issues that indicate insufficient system testing (TOB-ZETA-1, TOB-ZETA-4, and TOB-ZETA-12). Additionally, TOB-ZETA-3 underscores that care should be taken to prioritize new admin messages when they are added in the system. Lastly, the system should adhere more closely to the documentation for the third-party libraries in use to avoid difficult-to-find mistakes like TOB-ZETA-7, which describes reuse of a protocol buffer field number.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that ZetaChain take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Address known design limitations and add defense-in-depth features.** Review and prioritize long-standing, security-relevant issues such as TOB-ZETA-11 that make the protocol more robust and easier to understand.
- **Perform property based testing.** The end-to-end testing harness should test pseudo-random sequences of actions and make assertions according to what is expected. This will help eliminate problematic gaps such as TOB-ZETA-12.
- **Improve the documentation.** The processing of cross-chain transactions warrants additional specification, in particular a detailed end-to-end flow and invariants.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	6
Informational	5
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	1
Auditing and Logging	1
Data Validation	6
Error Reporting	1
Patching	1
Undefined Behavior	2

Project Goals

The engagement was scoped to provide a security assessment of the ZetaChain's Node updates. Specifically, we sought to answer the following non-exhaustive list of questions:

- Is the new authorization implementation sound?
- Are the messages guarded by an appropriate authorization check?
- Does the upgrade to a new Cosmos SDK version follow the official guidelines?
- Did the refactoring of the cross-chain message processing cause regressions?
- Can processing of recursive events cause stack overflows?

Project Targets

The engagement involved a review and testing of the following target.

Node

Repository	https://github.com/zeta-chain/node
Version	`5e9068a90e1480525467cb254fe36302c14f84ec`
Type	Cosmos-SDK, Go
Platform	Linux, MacOS

Project Coverage

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

- PR #2100 updates the Cosmos SDK to the 0.47.10 version. We reviewed this PR to assess whether the official upgrade guidelines are followed and particularly if the storage is upgraded correctly.
- PR #2289 and #2313 add a new method to validate if a message's caller is authorized by using an authorization list that contains various policies. We reviewed whether the authorization list is correctly validated when updated, and whether the new `CheckAuthorization` function correctly enforces the authorization process and is applied to all the messages that need to be guarded.
- PR #2317 and #2340 refactor the CCTX inbound and outbound validation logic. We reviewed possible introductions of issues such as double-spend and whether the new logic has the same behavior as the old one.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The protocol's smart contracts
- Code unrelated to the changes in scope

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Appendix C

Summary of Findings

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

ID	Title	Type	Severity
1	Missing validation of AuthorizationList at genesis and migration time	Data Validation	**Low**
2	VoteTSS does not check if the keygen status is failed	Data Validation	**Informational**
3	Missing transaction prioritization for several admin messages	Access Controls	**Low**
4	Updating observer set permits duplicates and inhibits slashing	Data Validation	**Low**
5	Median gas price is incorrect for odd-length arrays	Data Validation	**Low**
6	Events are emitted incorrectly or not at all	Auditing and Logging	**Informational**
7	Reuse of protocol buffers field number	Undefined Behavior	**Informational**
8	Use of unmaintained protocol buffers fork	Patching	**Informational**
9	Invalid address fields may be truncated and processed in error	Data Validation	**Low**
10	Unhandled error may result in incorrect query results for rate limiting and queries	Error Reporting	**Low**
11	Suggested improvements to CCTX identifier	Data Validation	**Informational**
12	Lack of tests for events emitted during ZEVM deposits	Undefined Behavior	**Undetermined**

Detailed Findings

1. Missing validation of AuthorizationList at genesis and migration time

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ZETA-1
Target: `x/authority/types/genesis.go`	

Description

The `GenesisState` has an `AuthorizationList` field; however, its `Validate` function is not called in the `GenesisState`'s `Validate` function (figure 1.1), nor when migrating the storage (figure 1.2).

```
// DefaultGenesis returns the default authority genesis state
func DefaultGenesis() *GenesisState {
	return &GenesisState{
		Policies:          DefaultPolicies(),
		ChainInfo:         DefaultChainInfo(),
		AuthorizationList: DefaultAuthorizationsList(),
	}
}


// Validate performs basic genesis state validation returning an error upon any
failure
func (gs GenesisState) Validate() error {
	if err := gs.Policies.Validate(); err != nil {
		return err
	}


	return gs.ChainInfo.Validate()
}
```

Figure 1.1: `GenesisState` (`x/authority/types/genesis.go`)

```
func MigrateStore(
	ctx sdk.Context,
	keeper authorityKeeper,
) error {
	keeper.SetAuthorizationList(ctx, types.DefaultAuthorizationsList())
	return nil
}
```

Figure 1.2: `MigrateStore` function (`x/authority/migrations/v2/migrate.go`)

An `AuthorizationList` field keeps a list of `Authorization` values where each has a policy (defining who is authorized) and the corresponding message URL that can be called. The `AuthorizationList`'s `Validate` function checks that no message URLs are duplicated in the list (i.e., that the same message URL is not listed twice with different policies).

Exploit Scenario

Alice sets the default authorization list with three duplicates of the same message URL. The system is deployed. Alice then wants to add a new authorization, but the transaction unexpectedly fails. She tried to remove one message URL, but the transaction fails again because two instances remain and the `Validate` function fails. The system can not add or remove any authorization.

Recommendations

Short term, call the `gs.AuthorizationList.Validate` function in the `GenesisState`'s `Validate` function.

Long term, add tests to validate that the default value of a possible item, such as an `AuthorizationList`, is valid by testing the system with both a valid and invalid `AuthorizationList`.

2. VoteTSS does not check if the keygen status is failed

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ZETA-2
Target: `x/observer/keeper/msg_server_vote_tss.go`	

Description

The `VoteTSS` function does not check if the keygen state is set to failed and does not return if it is. As a result, if the node calling the function did not yet vote, its vote will be added to the ballot, the ballot's final status cannot change, and the function will return when calling the `CheckIfFinalizingVote` function because the `isFinalized` function will be false.

```
// Fails if the keygen does not exist, the keygen has been already
// completed, or the keygen has failed.
//
// Only node accounts are authorized to broadcast this message.
func (k msgServer) VoteTSS(goCtx context.Context, msg *types.MsgVoteTSS)
(*types.MsgVoteTSSResponse, error) {
	ctx := sdk.UnwrapSDKContext(goCtx)

	// Checks whether a signer is authorized to sign, by checking their address
against the observer mapper
	// which contains the observer list for the chain and type.
	_, found := k.GetNodeAccount(ctx, msg.Creator)
	if !found {
		return nil, errorsmod.Wrapf(
			sdkerrors.ErrorInvalidSigner,
			"%s, signer %s does not have a node account set", voteTSSid,
msg.Creator)
	}

	// No need to create a ballot if keygen does not exist.
	keygen, found := k.GetKeygen(ctx)
	if !found {
		return &types.MsgVoteTSSResponse{},
errorsmod.Wrap(types.ErrKeygenNotFound, voteTSSid)
	}

	// Use a separate transaction to update keygen status to pending when trying
to change the TSS address.
	if keygen.Status == types.KeygenStatus_KeyGenSuccess {
		return &types.MsgVoteTSSResponse{},
errorsmod.Wrap(types.ErrKeygenCompleted, voteTSSid)
	}
```

```
...
	ballot, isFinalized := k.CheckIfFinalizingVote(ctx, ballot)
if !isFinalized {
	return &types.MsgVoteTSSResponse{
		VoteFinalized: isFinalized,
		BallotCreated: ballotCreated,
		KeygenSuccess: false,
	}, nil
}
...
```

Figure 2.1: Snippet of the `VoteTSS` function (`x/observer/keeper/msg_server_vote_tss.go`)

Recommendations

Short term, add a check and return an error if the `keygen.Status` is equal to `KeygenStatus_KeyGenFailed`.

Long term, improve the unit tests by having a happy and unhappy test for each necessary precondition. In this case, calling the `VoteTSS` function when the `keygen.Status` is `KeygenStatus_KeyGenFailed` should fail.

3. Missing transaction prioritization for several admin messages

Severity: **Low**	Difficulty: **Medium**
Type: Access Controls	Finding ID: TOB-ZETA-3
Target: `node/app/ante/ante.go`	

Description

For some system (privileged) messages, ZetaChain uses a special ante handler that sets the messages' priority above unprivileged users, ensuring that system transactions are processed without delay. However, several time-sensitive message types, such as those in the `EmergencyPolicyMessages` group, lack prioritization and should be prioritized (among the others highlighted in figure 3.3).

```
if IsSystemTx(tx, isAuthorized) {
	anteHandler = newCosmosAnteHandlerForSystemTx(options)
}
```

Figure 3.1: Special ante handler weights system transactions above others (node/app/ante/ante.go#108–111)

```
case *crosschaintypes.MsgVoteGasPrice,
	*crosschaintypes.MsgVoteOutbound,
	*crosschaintypes.MsgVoteInbound,
	*crosschaintypes.MsgAddOutboundTracker,
	*crosschaintypes.MsgAddInboundTracker,
	*observertypes.MsgVoteBlockHeader,
	*observertypes.MsgVoteTSS,
	*observertypes.MsgVoteBlame:
```

Figure 3.2: Messages considered system transactions (node/app/ante/ante.go#172–179)

```
var (
	// OperationPolicyMessages keeps track of the message URLs that can, by
default, only be executed by operational policy address
	OperationPolicyMessages = []string{
		"/zetachain.zetacore.crosschain.MsgRefundAbortedCCTX",
		"/zetachain.zetacore.crosschain.MsgAbortStuckCCTX",
		"/zetachain.zetacore.crosschain.MsgUpdateRateLimiterFlags",
		"/zetachain.zetacore.fungible.MsgDeploySystemContracts",
		"/zetachain.zetacore.fungible.MsgUpdateZRC20LiquidityCap",
		"/zetachain.zetacore.fungible.MsgUpdateZRC20WithdrawFee",
		"/zetachain.zetacore.fungible.MsgUnpauseZRC20",
		"/zetachain.zetacore.observer.MsgResetChainNonces",
```

```
                "/zetachain.zetacore.observer.MsgUpdateChainParams",
                "/zetachain.zetacore.observer.MsgEnableCCTX",
                "/zetachain.zetacore.observer.MsgUpdateGasPriceIncreaseFlags",
        }
        // AdminPolicyMessages keeps track of the message URLs that can, by default,
only be executed by admin policy address
        AdminPolicyMessages = []string{
                "/zetachain.zetacore.crosschain.MsgMigrateTssFunds",
                "/zetachain.zetacore.crosschain.MsgUpdateTssAddress",
                "/zetachain.zetacore.crosschain.MsgWhitelistERC20",
                "/zetachain.zetacore.fungible.MsgUpdateContractBytecode",
                "/zetachain.zetacore.fungible.MsgUpdateSystemContract",
                "/zetachain.zetacore.fungible.MsgRemoveForeignCoin",
                "/zetachain.zetacore.fungible.MsgDeployFungibleCoinZRC20",
                "/zetachain.zetacore.observer.MsgUpdateObserver",
                "/zetachain.zetacore.observer.MsgAddObserver",
                "/zetachain.zetacore.observer.MsgRemoveChainParams",
                "/zetachain.zetacore.authority.MsgAddAuthorization",
                "/zetachain.zetacore.authority.MsgRemoveAuthorization",
                "/zetachain.zetacore.authority.MsgUpdateChainInfo",
                "/zetachain.zetacore.lightclient.MsgEnableHeaderVerification",
        }
        // EmergencyPolicyMessages keeps track of the message URLs that can, by
default, only be executed by emergency policy address
        EmergencyPolicyMessages = []string{
                "/zetachain.zetacore.crosschain.MsgAddInboundTracker",
                "/zetachain.zetacore.crosschain.MsgAddOutboundTracker",
                "/zetachain.zetacore.crosschain.MsgRemoveOutboundTracker",
                "/zetachain.zetacore.fungible.MsgPauseZRC20",
                "/zetachain.zetacore.observer.MsgUpdateKeygen",
                "/zetachain.zetacore.observer.MsgDisableCCTX",
                "/zetachain.zetacore.lightclient.MsgDisableHeaderVerification",
        }
)
```

Figure 3.3: Complete list of (node/x/authority/types/authorization_list.go#9–51)

Exploit Scenario

To mitigate a security incident, the emergency group takes action to pause cross-chain transactions. However, the transaction-ordering issue causes some of an attacker's transactions to be processed before pausing takes effect.

Recommendations

Short term, review the message types that require a privileged role, and prioritize them unless there is a good reason not to.

Long term, create automated processes that are run in CI to keep admin defaults up to date as new messages are added and roles are changed.

4. Updating observer set permits duplicates and inhibits slashing

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ZETA-4
Target: `x/observer/keeper/observer_set.go`	

Description

ZetaChain has message types to add, remove, and update the observer set that is responsible for voting on cross-chain transactions' finality until a given threshold is reached. The message to add an observer ensures that duplicates are not added to the observer set, but this is not checked when the update message is used, allowing an existing observer to be duplicated in the set. While this does not appear to give them more voting power, it does prevent the slashing process from atomically removing them. It would require admin intervention to fully remove the observer's duplicate entry by updating the retained entry once more.

```
func (k Keeper) UpdateObserverAddress(ctx sdk.Context, oldObserverAddress,
newObserverAddress string) error {
	observerSet, found := k.GetObserverSet(ctx)
	if !found {
		return types.ErrObserverSetNotFound
	}
	for i, addr := range observerSet.ObserverList {
		if addr == oldObserverAddress {
			observerSet.ObserverList[i] = newObserverAddress
			k.SetObserverSet(ctx, observerSet)
			return nil
		}
	}
	return types.ErrUpdateObserver
}
```

Figure 4.1: Replacing an old observer does not ensure the new observer is not already one (`audit-zetachain-go/x/observer/keeper/observer_set.go#70–83`)

Exploit Scenario

An observer is incidentally duplicated while updating observers. This observer is later slashed but not fully removed from the observer set. Admins must coordinate to fully remove the observer from its role.

Recommendations

Short term, check that the “new” observer replacing the old one is not already a member of the observer set.

Long term, perform property-based testing and check for properties such as the uniqueness of the observer set. Consider deprecating the update functionality, as it is redundant given the existing functionality to add and remove observers.

5. Median gas price is incorrect for odd-length arrays

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ZETA-5
Target: `x/crosschain/keeper/msg_server_vote_gas_price.go`	

Description

When ZetaChain observers vote on the gas price, the median of their votes is used; however, the median is computed incorrectly when the length of the array of values is odd. If the array is an odd length, the average should be taken against the two middle values:

$$(array[(len(array) \,/\, 2)] \;+\; array[(len(array) \,/\, 2) \;+\; 1]) \,/\, 2$$

```
func medianOfArray(values []uint64) int {
	array := make([]indexValue, len(values))
	for i, v := range values {
		array[i] = indexValue{Index: i, Value: v}
	}
	sort.SliceStable(array, func(i, j int) bool {
		return array[i].Value < array[j].Value
	})
	l := len(array)
	return array[l/2].Index
}
```

Figure 5.1: Median value for odd-length arrays is less than expected (audit-zetachain-go/x/crosschain/keeper/msg_server_vote_gas_price.go#111–121)

Exploit Scenario

An odd number of observers vote for the gas price. This causes the gas price to be lower than expected, since the value computed is lower than the average of the middle values.

Recommendations

Short term, use the correct formula for odd-length arrays to compute the median.

Long term, perform testing for edge cases of mathematical formulas and use implementations of other libraries that should be equivalent for differential testing.

6. Events are emitted incorrectly or not at all

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-ZETA-6
Target: `x/observer/keeper/msg_server_vote_block_header.go`, `x/observer/keeper/msg_server_add_observer.go`	

Description

The `VoteBlockHeader` function does not emit the `BallotCreated` event when a new ballot is created and `isNew` is true (figure 6.1), while the `AddObserver` function emits the `AddObserver` event even when the observer was already part of the set (figure 6.2).

```
func (k msgServer) VoteBlockHeader(
	goCtx context.Context,
	msg *types.MsgVoteBlockHeader,
) (*types.MsgVoteBlockHeaderResponse, error) {
	...
	_, isFinalized, isNew, err := k.VoteOnBallot(
		ctx,
		chain,
		msg.Digest(),
		types.ObservationType_InboundTx,
		msg.Creator,
		types.VoteType_SuccessObservation,
	)
	if err != nil {
		return nil, sdkerrors.Wrap(err, voteBlockHeaderID)
	}

	if !isFinalized {
		return &types.MsgVoteBlockHeaderResponse{
			BallotCreated: isNew,
			VoteFinalized: false,
		}, nil
	}
	...
```

Figure 6.1: Snippet of the VoteBlockHeader function (x/observer/keeper/msg_server_vote_block_header.go#L16-L68)

```
func (k msgServer) AddObserver(
	goCtx context.Context,
	msg *types.MsgAddObserver,
) (*types.MsgAddObserverResponse, error) {
	...
	k.AddObserverToSet(ctx, msg.ObserverAddress)
```

```
	observerSet, _ := k.GetObserverSet(ctx)

	k.SetLastObserverCount(ctx, &types.LastObserverCount{Count:
observerSet.LenUint()})
	EmitEventAddObserver(
		ctx,
		observerSet.LenUint(),
		msg.ObserverAddress,
		granteeAddress.String(),
		msg.ZetaclientGranteePubkey,
	)
	...
```

Figure 6.2: Snippet of the `AddObserver` function (`x/observer/keeper/msg_server_add_observer.go#L17-L68`)

Recommendations

Short term, emit the `BallotCreated` event in the `VoteBlockHeader` function when `isNew` is true. Make the `AddObserverToSet` function returns a Boolean value indicating if the observer was added to the set, and emit the event only in that case.

7. Reuse of protocol buffers field number

Severity: **Informational**	Difficulty: **Undetermined**
Type: Undefined Behavior	Finding ID: TOB-ZETA-7
Target: `proto/zetachain/zetacore/authority/genesis.proto`	

Description

The new `GenesisState` message (figure 7.2) reuses the field number 2 for the new `authorization_list` field with the old `chain_info` field now having a field number of 3.

```
message GenesisState {
  Policies policies = 1 [ (gogoproto.nullable) = false ];
  ChainInfo chain_info = 2 [ (gogoproto.nullable) = false ];
}
```

Figure 7.1: Old `GenesisState` message

```
message GenesisState {
  Policies policies = 1 [ (gogoproto.nullable) = false ];
  AuthorizationList authorization_list = 2 [ (gogoproto.nullable) = false ];
  ChainInfo chain_info = 3 [ (gogoproto.nullable) = false ];
}
```

Figure 7.2: New `GenesisState` message

The protocol buffers documentation explains that reusing the field number causes the decoding of messages to be ambiguous and should be avoided. However, we did not find that this caused the error, so this issue is informational in severity.

Recommendations

Short term, add the new `authorizan_list` field at the end with a field number 3 and leave the `chain_info` with the field number 2.

Long term, when modifying a protocol buffers message either by adding or removing a field, never reuse a field number; instead in the former case, use a new field number, and in the latter, reserve the field number so it cannot be reused.

8. Use of unmaintained protocol buffers fork

Severity: **Informational**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-ZETA-8
Target: `go.mod`	

Description

The Comsos v0.47.10 SDK upgrade guide instructs developers to migrate to github.com/cosmos/gogoproto and remove the replace directive from the `go.mod` file. However, the ZetaChain project still uses the replace directive and has yet to update.

```
github.com/gogo/protobuf => github.com/regen-network/protobuf v1.3.3-alpha.regen.1
```

Figure 8.1: Go module's dependencies (go.mod#354)

Recommendations

Short term, remove the replace directive and depend directly on github.com/cosmos/gogoproto.

Long term, carefully follow every upgrade guide, and run vulnerability scanners like govulncheck and outdated dependency detection tools in CI.

9. Invalid address fields may be truncated and processed in error

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ZETA-9
Target: Cross-chain and fungible modules	

Description

Several message types, including `MsgVoteInbound`, `MsgUpdateSystemContract`, and `MsgWhitelistERC20`, have fields that should be valid Ethereum addresses (i.e., 20 byte and hexadecimal), but they are not validated before they are converted from strings. This lack of validation could truncate the data and enable the use of invalid data that should be rejected.

Instead, these fields should be rejected in the messages' `ValidateBasic` implementation if they do not pass the checks performed by go-ethereum's `IsHexAddress` method. We have provided a Semgrep rule, "hex2address," in appendix C to identify this issue, and we recommend running the rule in CI.

```
newSystemContractAddr := ethcommon.HexToAddress(msg.NewSystemContractAddress)
```

Figure 9.1: `MsgUpdateSystemContract`'s `NewSystemContractAddress` field cast without validation (`x/fungible/keeper/msg_server_update_system_contract.go#29`)

```
if ethcommon.HexToAddress(msg.Erc20Address) == (ethcommon.Address{}) {
```

Figure 9.2: `MsgWhitelistERC20`'s `Erc20Addresss` field cast without validation (`x/crosschain/types/message_whitelist_erc20.go#57`)

```
to := ethcommon.HexToAddress(cctx.GetCurrentOutboundParam().Receiver)
```

Figure 9.3: `MsgVoteInbound`'s `Receiver` field cast without validation (`x/crosschain/keeper/evm_deposit.go#27`)

Exploit Scenario

The processing of invalid inbound votes or ERC20 allowlist operations obscures issues in observers and makes troubleshooting difficult, causing downtime.

Recommendations

Short term, check that strings are valid Ethereum addresses before using a method that truncates without warning.

Long term, review messages for overly permissive data validation. Make data validation stricter and implement regression tests.

10. Unhandled error may result in incorrect query results for rate limiting and queries

Severity: **Low**	Difficulty: **Medium**
Type: Error Reporting	Finding ID: TOB-ZETA-10
Target: `x/crosschain/keeper/`	

Description

When cross-chain transactions are created, they are validated with `ValidateInbound` and then persisted with `SetCctxAndNonceToCctxAndInboundHashToCctx`. However, if the TSS is not available in the setter method, `SetCctxAndNonceToCctxAndInboundHashToCctx` will return early and not record the data used by other functions, such as the rater limiter query. Any cross-chain transaction that hits this unhandled error case will thus not be reflected in the rate limiter and affect the `zetaclient`'s processing of cross-chain transactions.

```
// ValidateInbound is the only entry-point to create new CCTX (eg. when observers
voting is done or new inbound event is detected).
// It creates new CCTX object and calls InitiateOutbound method.
func (k Keeper) ValidateInbound(
	ctx sdk.Context,
	msg *types.MsgVoteInbound,
	shouldPayGas bool,
) (*types.CrossChainTx, error) {
[...]
	_, err = k.InitiateOutbound(ctx, InitiateOutboundConfig{
		CCTX:         &cctx,
		ShouldPayGas: shouldPayGas,
	})
[...]
	k.SetCctxAndNonceToCctxAndInboundHashToCctx(ctx, cctx)

	return &cctx, nil
}
```

Figure 10.1: `ValidateInbound` does not fail if `SetCctxAndNonceToCctxAndInboundHashToCctx` returns early (`x/crosschain/keeper/cctx_orchestrator_validate_inbound.go#12–57`)

```
func (k Keeper) SetCctxAndNonceToCctxAndInboundHashToCctx(ctx sdk.Context, cctx
types.CrossChainTx) {
	tss, found := k.zetaObserverKeeper.GetTSS(ctx)
```

```
        if !found {
                return
        }
```

Figure 10.2: Record keeping of CCTX is skipped if TSS is unavailable (x/crosschain/keeper/cctx.go#19–23)

```
cctx, err := getCctxByChainIDAndNonce(k, ctx, tss.TssPubkey, chain.ChainId, nonce)
```

Figure 10.3: `RateLimiterInput` relies on this value being available (x/crosschain/keeper/grpc_query_cctx_rate_limit.go#133)

Exploit Scenario

The rate limiter returns an incorrect value and causes the `zetaclient` to stall or prematurely process transactions.

Recommendations

Short term, return an error if `SetCctxAndNonceToCctxAndInboundHashToCctx` is unsuccessful and handle it at every callsite.

Long term, exhaustively specify what is expected if errors occur, and do not leave errors unhandled.

11. Suggested improvements to CCTX identifier

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ZETA-11
Target: Cross-chain transaction processing	

Description

For events emitted during deposits to ZetaChain's EVM (ZEVM), a different identifier overrides the `observedHash` than the one that is used for inbound cross-chain transactions (figures 11.1 and 11.2). However, this special treatment, and how it is identified by checking whether a type cast succeeds, is error prone. Ideally, the cross-chain transaction identifier would be computed the same for EVM transactions irrespective of how many events they emit. Additionally, this identifier should be versioned to prevent the replay of old cross-chain transactions or re-interpreting the message.

```
ctx = ctx.WithValue(InCCTXIndexKey, cctx.Index)
txOrigin := cctx.InboundParams.TxOrigin
if txOrigin == "" {
	txOrigin = inboundSender
}

err = k.ProcessLogs(ctx, logs, to, txOrigin)
if err != nil {
	// ProcessLogs should not error; error indicates exception, should abort
	return false, errors.Wrap(types.ErrCannotProcessWithdrawal, err.Error())
}
```

Figure 11.1: CCTX identifier for event emitted in EVM deposit (x/crosschain/keeper/evm_deposit.go#107–117)

```
inCctxIndex, ok := ctx.Value(InCCTXIndexKey).(string)
if ok {
	cctx.InboundParams.ObservedHash = inCctxIndex
}
```

Figure 11.2: Type casting used to decide whether to override `ObservedHash` (x/crosschain/keeper/cctx_orchestrator_validate_inbound.go#50–53)

Given that `SaveObservedInboundInformation` and `IsFinalizedInbound` already require the `observedHash`/`inboundHash`, the event index, and sender chain ID (figures 11.3 and 11.4), it would make sense to use these values with proper domain separation to

compute a unique CCTX identifier in place of the current inconsistent scheme. Including the address that emitted the event and an automatically incrementing ID would further strengthen the scheme and make it straightforward to create indexers that process events from only authorized addresses.

```
func (k Keeper) SaveObservedInboundInformation(ctx sdk.Context, cctx
*types.CrossChainTx, eventIndex uint64) {
	EmitEventInboundFinalized(ctx, cctx)
	k.AddFinalizedInbound(ctx,
		cctx.GetInboundParams().ObservedHash,
		cctx.GetInboundParams().SenderChainId,
		eventIndex)
```

Figure 11.3: Newly finalized CCTX's status is recorded (x/crosschain/keeper/msg_server_vote_inbound_tx.go#119–124)

```
if k.IsFinalizedInbound(tmpCtx, msg.InboundHash, msg.SenderChainId, msg.EventIndex)
```

Figure 11.4: Validation that a CCTX has not already been processed (x/crosschain/keeper/msg_server_vote_inbound_tx.go#84)

Recommendations

Short term, use a consistent, versioned, domain-separated hashing scheme like EIP-712, and store the CCTX identifier as a 32-byte byte array instead of a string.

Long term, avoid making incremental, untested changes that complicate the code without addressing the root cause (e.g., PR#1372).

12. Lack of tests for events emitted during ZEVM deposits

Severity: **Undetermined**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-ZETA-12
Target: `x/crosschain/keeper/`	

Description

When ZetaChain processes deposits, `ZRC20Withdrawal` and `ZetaConnectorZEVMZetaSent` events may be emitted. In the current implementation, `InitiateOutbound` can recursively call itself indirectly when `HandleEVMDeposit` calls `ProcessLogs` (see appendix D). Additionally, Zeta send events can originate and target ZetaChain itself, potentially causing send events to be emitted recursively. As advised in our April review, there does not seem to be a valid reason to support cross-chain messages where the origin and destination chain ID are the same, since normal ERC20 operations can be performed natively on the EVM and without requiring bridging functionality. While we have not yet confirmed that a stack overflow—and thus halting the chain—is possible, the use of recursion should be eliminated as a defense-in-depth measure.

```
func (c CCTXGatewayZEVM) InitiateOutbound(
       ctx sdk.Context,
       config InitiateOutboundConfig,
) (newCCTXStatus types.CctxStatus, err error) {
       tmpCtx, commit := ctx.CacheContext()
       isContractReverted, err := c.crosschainKeeper.HandleEVMDeposit(tmpCtx,
config.CCTX)
[...]
```

Figure 12.1: Handling of EVM deposits (x/crosschain/keeper/cctx_gateway_zevm.go#22–27)

This method is especially a concern if it is chosen to process events for inbound Zeta deposits; this is not currently done, but appears to cause infinite loops (see appendix D). This attack vector should be investigated further for inbound ERC20 deposits because we were not yet able to rule out how the events emitted by the `onCrossChainCall` hook are processed.

```
func (k Keeper) HandleEVMDeposit(ctx sdk.Context, cctx *types.CrossChainTx) (bool,
error) {
[...]
       if inboundCoinType == coin.CoinType_Zeta {
```

```
[...]
            // if coin type is Zeta, this is a deposit ZETA to zEVM cctx.
            evmTxResponse, err := k.fungibleKeeper.ZETADepositAndCallContract(
                  ctx,
                  sender,
                  to,
                  inboundSenderChainID,
                  inboundAmount,
                  data,
                  indexBytes,
            )
[...]
```

Figure 12.2: Inbound deposits of Zeta do not currently process logs (x/crosschain/keeper/evm_deposit.go#26–65)

Furthermore, processing of events in cross-chain messages is largely untested, and there are gaps for the following cases:

- Withdraw ZRC20 in `onMessage` call
- Send Zeta in `onMessage` call
- Withdraw ZRC20 in `onCrosschainCall` call
- Send Zeta in `onCrosschainCall` call

We recommend implementing these tests (including the case where the origin and destination chain ID are the same) and specifying what is expected to be supported and how the events should behave.

For example, changing an existing test to create Zeta deposits from ZetaChain to ZetaChain appears to crash the `getTransactionReceipt` RPC endpoint (which is then recovered). Figure 12.3 demonstrates running a test command after applying the patch in appendix D.1 without modifying the `TestDappNoRevert` contract:

```
go run ./cmd/zetae2e/ run
message_passing_zevm_to_evm_revert_fail:10000000000000000006 --config
cmd/zetae2e/config/local.yml --verbose
```

Figure 12.3: Test command to reproduce RPC crash (recoverable)

```
9:23PM ERR RPC method eth_getTransactionReceipt crashed: runtime error: index out of
range [1] with length 1
goroutine 11335 [running]:
github.com/ethereum/go-ethereum/rpc.(*callback).call.func1()
        /go/pkg/mod/github.com/ethereum/go-ethereum@v1.10.26/rpc/service.go:200
+0x74
panic({0x2f23be0?, 0x40038cf788?})
        /usr/local/go/src/runtime/panic.go:770 +0x124
github.com/zeta-chain/zetacore/rpc/types.(*ParsedTxs).updateTx(0x4003222700, 0x1,
```

```
{0x400296ab40?, 0x28ad2f1e5ffde1e6?, 0x5ace28441f82a67a?})
        /go/delivery/zeta-node/rpc/types/events.go:334 +0x160
github.com/zeta-chain/zetacore/rpc/types.ParseTxResult
```

Figure 12.4: `getTransactionReceipt` RPC method crashing

Exploit Scenario

Events are recursively emitted in deposits and create a call stack that exceeds the host's stack bound, causing nodes to crash and halting the chain's progress.

Recommendations

Short term, remove recursion by processing the events pushed to a stack in a loop, and perform extensive testing of events emitted in the hooks called by the ZEVM connector.

Long term, design and implement features methodically and identify potential abuses that need to be mitigated prior to merging the feature.

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 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	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions 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.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

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

B. Non-Security-Related Findings

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities, become easily exploitable in future releases, or decrease code readability. We recommend fixing the issues reported here.

- The following comment is misleading and does not reflect the changes made in PR#1372. It should be removed or updated.

```
// Bump gasLimit by event index (which is very unlikely to be larger than 1000) to
always have different ZetaSent events msgs.
```

Figure B.1: Outdated comment (x/crosschain/keeper/evm_hooks.go#244)

- Instead of requiring that developers remember to call `Validate` prior to setting the `AuthorizationList`, it could always be called in `SetAuthorization` to prevent issues like (TOB-ZETA-1).

```
func (a *AuthorizationList) SetAuthorization(authorization Authorization) {
	for i, auth := range a.Authorizations {
		if auth.MsgUrl == authorization.MsgUrl {
			a.Authorizations[i].AuthorizedPolicy =
authorization.AuthorizedPolicy
			return
		}
	}
	a.Authorizations = append(a.Authorizations, authorization)
}
```

Figure B.2: Setter that does not check input is valid (x/authority/types/authorization_list.go#89–97)

- This should be moved to the calling function, VoteInbound, as it performs a similar check for `isNew` and performs additional error handling and would make it more clear the event is only emitted when the ballot is successfully created.

```
if isNew {
	EmitEventBallotCreated(ctx, ballot, inboundHash, senderChain.String())
}
```

Figure B.3: Emitting of event hidden in nested call before error handling is completed (x/observer/keeper/vote_inbound.go#75–77)

- This `SetlastObserverCount` call does not set the `LastChangeHeight` field of `LastObserverCount` struct while in all other calls it is set. It seems that

`LastChangeHeight` field is never used anywhere so consider if it should be removed.

```
k.SetLastObserverCount(ctx, &types.LastObserverCount{Count: observerSet.LenUint()})
```

Figure B.4: `SetLastObserverCount` function call (x/observer/keeper/vote_inbound.go#75–77)

C. Semgrep Rules for ZetaChain

During our review, we use the Semgrep static analysis tool to write custom lints for ZetaChain in order to detect improper use of Cosmos SDK and go-ethereum APIs. The "hex2address" rule resulted in TOB-ZETA-9.

- The following rule, "wrong-context," detects the use of a temporary context after committing/writing the Cosmos SDK's CacheContext when it should likely be using the context that has been persisted.

```
rules:
  - id: wrong-context
    patterns:
      - pattern: |
          $TMP, $WRITE := $O.CacheContext()
          ...
          $WRITE()
          ...
          <... $TMP ...>
    message: "Usage of $TMP after $WRITE is called"
    languages: [go]
    severity: WARNING
```

Figure C.1: "wrong-context" Semgrep rule

- The following rule, "hex2address," detects converting a string to an address, truncating the length without error as opposed to validating that the string is 20 bytes as Ethereum addresses should be.

```
rules:
  - id: hex2address
    patterns:
      - pattern: |
          ethcommon.HexToAddress($X)
      - pattern-not: |
          if !ethcommon.HexToAddress($X) {
            ...
          }
          ...
          ethcommon.HexToAddress($X)
      - pattern-not-inside: |
          if ethcommon.HexToAddress($X) == (ethcommon.Address{}) {
            ...
          }
    message: Truncating address conversion without validation
    languages:
      - go
    severity: WARNING
```

Figure C.2: "hex2address" Semgrep rule

D. Testing Recursive Processing of Logs

During our investigation of whether a recursive process of events can cause stack overflows (TOB-ZETA-12), we tested modifications to the test files and node code that follow.

```
diff --git a/e2e/e2etests/test_message_passing_zevm_to_evm_revert_fail.go
b/e2e/e2etests/test_message_passing_zevm_to_evm_revert_fail.go
index cc22db23..de02b224 100644
--- a/e2e/e2etests/test_message_passing_zevm_to_evm_revert_fail.go
+++ b/e2e/e2etests/test_message_passing_zevm_to_evm_revert_fail.go
@@ -34,10 +34,10 @@ func TestMessagePassingZEVMtoEVMRevertFail(r *runner.E2ERunner,
args []string) {
       utils.RequireTxSuccessful(r, receipt)

       // Set destination details
-      EVMChainID, err := r.EVMClient.ChainID(r.Ctx)
-      require.NoError(r, err)
+      // EVMChainID, err := r.EVMClient.ChainID(r.Ctx)
+      // require.NoError(r, err)

-      destinationAddress := r.EvmTestDAppAddr
+      // destinationAddress := r.EvmTestDAppAddr

       // Contract call originates from ZEVM chain
       r.ZEVMAuth.Value = amount
@@ -63,11 +63,14 @@ func TestMessagePassingZEVMtoEVMRevertFail(r *runner.E2ERunner,
args []string) {
       previousBalanceZEVM, err := r.WZeta.BalanceOf(&bind.CallOpts{},
testDappNoRevertAddr)
       require.NoError(r, err)

+      zEVMChainID, err := r.ZEVMClient.ChainID(r.Ctx)
+      require.NoError(r, err)
+
       // Send message with doRevert
-      tx, err = testDappNoRevert.SendHelloWorld(r.ZEVMAuth, destinationAddress,
EVMChainID, amount, true)
+      tx, err = testDappNoRevert.SendHelloWorld(r.ZEVMAuth, testDappNoRevertAddr,
zEVMChainID, amount, false)
       require.NoError(r, err)

-      r.Logger.Info("TestDAppNoRevert.SendHello tx hash: %s", tx.Hash().Hex())
+      r.Logger.Info("TestDAppNoRevert.SendHelloWorld tx hash: %s", tx.Hash().Hex())
       receipt = utils.MustWaitForTxReceipt(r.Ctx, r.ZEVMClient, tx, r.Logger,
r.ReceiptTimeout)
       utils.RequireTxSuccessful(r, receipt)
```

Figure D.1: Patch to cause cross-chain calls to same chain (e2e/e2etests/test_message_passing_zevm_to_evm_revert_fail.go)

```
diff --git a/x/crosschain/keeper/evm_deposit.go b/x/crosschain/keeper/evm_deposit.go
index b1e55e3f..cf92287c 100644
--- a/x/crosschain/keeper/evm_deposit.go
+++ b/x/crosschain/keeper/evm_deposit.go
@@ -63,6 +63,22 @@ func (k Keeper) HandleEVMDeposit(ctx sdk.Context, cctx
*types.CrossChainTx) (boo
                    data,
                    indexBytes,
             )
+            if !evmTxResponse.Failed() {
+                   logs := evmtypes.LogsToEthereum(evmTxResponse.Logs)
+                   if len(logs) > 0 {
+                          ctx = ctx.WithValue(InCCTXIndexKey, cctx.Index)
+                          txOrigin := cctx.InboundParams.TxOrigin
+                          if txOrigin == "" {
+                                 txOrigin = inboundSender
+                          }
+
+                          err = k.ProcessLogs(ctx, logs, to, txOrigin)
+                          if err != nil {
+                                 // ProcessLogs should not error; error indicates
exception, should abort
+                                 return false,
errors.Wrap(types.ErrCannotProcessWithdrawal, err.Error())
+                          }
+                   }
+            }
             if fungibletypes.IsContractReverted(evmTxResponse, err) ||
errShouldRevertCctx(err) {
                    return true, err
             } else if err != nil {
```

Figure D.2: Patch to process logs for inbound Zeta deposits (`x/crosschain/keeper/evm_deposit.go`)

```
diff --git a/e2e/contracts/testdappnorevert/TestDAppNoRevert.sol
b/e2e/contracts/testdappnorevert/TestDAppNoRevert.sol
index c7faef94..117f4a00 100644
--- a/e2e/contracts/testdappnorevert/TestDAppNoRevert.sol
+++ b/e2e/contracts/testdappnorevert/TestDAppNoRevert.sol
@@ -55,6 +55,7 @@ interface ZetaConnector {
 interface IERC20 {
     function transferFrom(address _from, address _to, uint256 _value) external
returns (bool success);
     function approve(address _spender, uint256 _value) external returns (bool
success);
+    function balanceOf(address account) external view returns (uint256);
 }

 /**
```

```
@@ -68,22 +69,35 @@ contract TestDAppNoRevert {
     error ErrorTransferringZeta();
     address public connector;
     address public zeta;
+    address public dest;
+    uint256 public chainId;
     constructor(address _connector, address _zetaToken) {
         connector = _connector;
         zeta = _zetaToken;
     }

     function onZetaMessage(ZetaInterfaces.ZetaMessage calldata zetaMessage)
external {
-        (, bool doRevert) = abi.decode(zetaMessage.message, (bytes32, bool));
-        require(doRevert == false,  "message says revert");
+        uint val =  IERC20(zeta).balanceOf(address(this));
+        bool success1 = IERC20(zeta).approve(address(connector), val);

-        emit HelloWorldEvent();
+        ZetaConnector(connector).send(
+            ZetaInterfaces.SendInput({
+                destinationChainId: chainId,
+                destinationAddress: abi.encodePacked(dest),
+                destinationGasLimit: 1_000_000,
+                message: abi.encode(HELLO_WORLD_MESSAGE_TYPE, false),
+                zetaValueAndGas: val,
+                zetaParams: abi.encode("")
+            })
+        );
     }

-    function sendHelloWorld(address destinationAddress, uint256 destinationChainId,
uint256 value, bool doRevert) external payable {
+    function sendHelloWorld(address destinationAddress, uint256 destinationChainId,
uint256 value, bool doRevert) public payable {
         bool success1 = IERC20(zeta).approve(address(connector), value);
         bool success2 = IERC20(zeta).transferFrom(msg.sender, address(this),
value);
         if (!(success1 && success2)) revert ErrorTransferringZeta();
+        chainId = block.chainid;
+        dest = address(this);

         ZetaConnector(connector).send(
             ZetaInterfaces.SendInput({
```

Figure D.3: Patch to create Zeta send events in onZetaMessage hook (e2e/contracts/testdappnorevert/TestDAppNoRevert.sol)

After applying the above diffs to the `message_passing_zevm_to_evm_revert_fail` test (figure D.1), `evm_deposit.go` (figure D.2), and `TestDappNoRevert.sol` (figure D.3), running the following command will result in an infinite loop of deposits and burns of Zeta:

```
go run ./cmd/zetae2e/ run
message_passing_zevm_to_evm_revert_fail:10000000000000000006 --config
cmd/zetae2e/config/local.yml --verbose
```

Figure D.4: Test command to cause infinite loop

We recommend removing the use of recursion to entirely mitigate the possibility of chain halts due to stack overflows or other denial-of-service vectors this issue enables.

E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On September 11, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the ZetaChain team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the issues described in this report, ZetaChain has resolved five issues, partially resolved one issue, and decided to not resolve six issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Missing validation of AuthorizationList at genesis and migration time	Resolved
2	VoteTSS does not check if the keygen status is failed	Resolved
3	Missing transaction prioritization for several admin messages	Unresolved
4	Updating observer set permits duplicates and inhibits slashing	Resolved
5	Median gas price is incorrect for odd-length arrays	Unresolved
6	Events are emitted incorrectly or not at all	Partially Resolved
7	Reuse of protocol buffers field number	Unresolved
8	Use of unmaintained protocol buffers fork	Unresolved
9	Invalid address fields may be truncated and processed in error	Resolved
10	Unhandled error may result in incorrect query results for rate limiting and queries	Resolved
11	Suggested improvements to CCTX identifier	Unresolved

12	Lack of tests for events emitted during ZEVM deposits	Unresolved

Detailed Fix Review Results

TOB-ZETA-1: Missing validation of AuthorizationList at genesis and migration time
Resolved in PR 2654. The `AuthorizationList` is validated in the `GenesisState`'s `Validate` function.

TOB-ZETA-2: VoteTSS does not check if the keygen status is failed
Resolved in PR 2674. The new logic allows voters to vote on older keygen ballots or ballots already finalized while disallowing the ballot from changing its status in these cases. This allows voters to get observer rewards for voting on a ballot.

TOB-ZETA-3: Missing transaction prioritization for several admin messages
Unresolved. The ZetaChain team gave the following response:

> *"These changes will be prioritized in a future release."*

TOB-ZETA-4: Updating observer set permits duplicates and inhibits slashing
Resolved in PR 2672. The `UpdateObserverAddress` function now checks that the new observer set is valid after the update.

TOB-ZETA-5: Median gas price is incorrect for odd-length arrays
Unresolved. The ZetaChain team gave the following response:

> *"The risk is minimal and we have a mechanism to update gas prices if the cctx is in a pending state for too long."*

TOB-ZETA-6: Events are emitted incorrectly or not at all
Partially resolved in PR 2672. The `AddObserver` function will not emit the event if an already present observer is to be added; instead, it returns an error. The `BallotCreated` event has not been added to the `VoteBlockHeader` function.

TOB-ZETA-7: Reuse of protocol buffers field number
Unresolved. The ZetaChain team gave the following response:

> *"Suggested improvement will be added to the backlog and may be addressed in a future release."*

TOB-ZETA-8: Use of unmaintained protocol buffers fork
Unresolved.

> "*Will be addressed in a future release*."

TOB-ZETA-9: Invalid address fields may be truncated and processed in error
Resolved in PR 2707. A new function has been added that validates that an address is valid for a certain chain; this function should always be used to validate the sender and receiver of a cross-chain transaction. It is currently used for the `MsgVoteInbound`, `MsgUpdateSystemContract`, and `MsgWhitelistERC20` messages. Note that, at the time of the fix review, the PR has not been merged yet; this will be done in the near future.

TOB-ZETA-10: Unhandled error may result in incorrect query results for rate limiting and queries
Resolved in PR 2748. The `SetCctxAndNonceToCctxAndInboundHashToCctx` function now takes the `tssPubkey` directly, and it is the job of the caller to get the `tssPubkey` and eventually handle the case where it is not present.

TOB-ZETA-11: Suggested improvements to CCTX identifier
Unresolved. The ZetaChain team gave the following response:

> *"Suggested improvement will be added to the backlog and may be addressed in a future release."*

TOB-ZETA-12: Lack of tests for events emitted during ZEVM deposits
Unresolved. The ZetaChain team gave the following response:

> *"We are making improvements to our tests across the whole codebase. This suggested improvement will be added to the backlog and may be addressed in a future release."*

F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.