



Sifchain - Continuous Audit Report

April 28, 2021

Cryptonics Consulting S.L.

Ramiro de Maeztu 7

46022 Valencia

SPAIN

<https://cryptonics.consulting/>

Table of Contents

Table of Contents	2
Disclaimer	4
Introduction	5
Purpose of this Report	5
Codebase Submitted for the Audit	5
Methodology	10
Functionality Overview	10
How to read this Report	11
Notes on Architecture / Functionality	12
Sifchain CLP	12
Peggy Ethereum Bridge	12
Sifchain CLP Findings	14
Summary	14
Code Quality Criteria	14
Detailed Findings	15
x/clp/keeper/Calculations.go: Token decimal adjustment is always executed	15
Limitations of pool addressing convention	15
Gas Limitations	16
Duplicate function implementations in calculations.go	16
Duplicate logic in calculations.go	16
Leftover TODO comments	17
Documentation refers to Thorchain	17
Peggy Ethereum Bridge Findings	18
Solidity Smart Contracts	18
Summary of Findings	18
Code Quality Criteria	19



Detailed Findings	20
Potential namespace collisions through token symbol	20
Potential for funds to be locked indefinitely in case the locked ERC20 tokens do not support returning values on transfer()	20
Potential for BridgeBank.lock() to always fail in case the it attempts to lock an ERC20 token that does not support returning values on transfer()	20
Missing checks for maximum consensus threshold	21
Malleable signatures	21
Duplicate validators are accepted in validator set	21
Ether transfers depending on gas stipend may not work if the recipient is a smart contract	22
Compiler version	22
Misleading behavior for recoverGas function	23
Bridge bank can receive ETH directly	23
Unnecessary modifier	23
Unnecessary protection in view function	24
Optimization on token deployment costs	24
Consider using a mock address for ETH	24
Incorrect natspect comments	24
Node package inclusion in Solidity files breaks build system	25
Provide formal documentation on why the number 30 Ethereum blocks was chosen as safe enough finality for Sifchain.	25
Go Components	26
Summary of Findings	26
Code Quality Criteria	26
Detailed Findings	27
Outdated indirect dependency with multiple known security vulnerabilities	27
Relayer quits silently on failure to connect to subscribe to Ethereum events	27
Missing default case in switch statement	28
Empty code blocks	28

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHORS AND THEIR EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS NOT A SECURITY WARRANTY, INVESTMENT ADVICE, OR AN ENDORSEMENT OF THE CLIENT OR ITS PRODUCTS. THIS AUDIT DOES NOT PROVIDE A SECURITY OR CORRECTNESS GUARANTEE OF THE AUDITED SOFTWARE.

Introduction

Purpose of this Report

Cryptonics Consulting has been engaged to perform a continuous security audit of Sifchain. This current audit report covers the implementation of the Sifchain **Blockchain node software**, including the liquidity pool implementation and the Peggy Ethereum bridge.

The objectives of the audit are as follows:

1. Determine the correct functioning of the system, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the code submitted in the following module in a public GitHub repository:

<https://github.com/Sifchain/sifnode/>

with the latest commit no: 12993563ecf61bea7462872bd14fd13861e3fe56

The following files are in scope for the audit process:

```
app
├── app.go
├── app_test.go
├── export.go
├── poolChange.go
├── prefix.go
├── release-20210501000000-AddDispensation.go
└── setupHandlers.go
```



```
cmd
├── ebrelayer
│   ├── contract
│   │   ├── abi.go
│   │   │   ├── bin
│   │   │   │   ├── BridgeBank
│   │   │   │   ├── BridgeRegistry
│   │   │   │   ├── CosmosBridge
│   │   │   │   ├── Oracle
│   │   │   │   └── Valset
│   │   └── bindings
│   │       ├── bridgebank
│   │       │   └── BridgeBank.go
│   │       ├── bridgeregistry
│   │       │   └── BridgeRegistry.go
│   │       ├── cosmosbridge
│   │       │   └── CosmosBridge.go
│   │       ├── oracle
│   │       │   └── Oracle.go
│   │       └── valset
│   │           └── Valset.go
│   ├── generator.go
│   └── types.go
├── main.go
├── relayer
│   ├── cosmos.go
│   ├── cosmos_test.go
│   ├── ethereum.go
│   ├── ethereum_test.go
│   ├── listMissedEvent.go
│   ├── network.go
│   └── network_test.go
├── replay.go
├── txs
│   ├── parser.go
│   ├── parser_test.go
│   ├── registry.go
│   ├── registry_test.go
│   ├── relayToCosmos.go
│   ├── relayToEthereum.go
│   ├── signature.go
│   ├── signature_test.go
│   ├── test_common.go
│   └── types.go
├── types
│   ├── ethEventBuffer.go
│   ├── events.go
│   └── types.go
├── sifcrg
│   ├── Dockerfile
│   └── main.go
├── sifgen
│   └── main.go
├── sifnodecli
│   └── main.go
└── sifnoded
    ├── genaccounts.go
    └── main.go
```



smart-contracts/contracts

- ├── BridgeBank
 - ├── BankStorage.sol
 - ├── BridgeBank.sol
 - ├── BridgeToken.sol
 - ├── CosmosBank.sol
 - ├── CosmosBankStorage.sol
 - ├── CosmosWhiteList.sol
 - ├── CosmosWhiteListStorage.sol
 - ├── EthereumBank.sol
 - ├── EthereumBankStorage.sol
 - ├── EthereumWhitelist.sol
 - ├── Pausable.sol
 - ├── PauserRole.sol
 - ├── SifchainTestToken.sol
 - └── ToLower.sol
- ├── BridgeRegistry.sol
- ├── CosmosBridge.sol
- ├── CosmosBridgeStorage.sol
- ├── Migrations.sol
- ├── MockUpgrade
 - └── MockCosmosBridgeUpgrade.sol
- ├── Oracle.sol
- ├── OracleStorage.sol
- ├── Valset.sol
- └── ValsetStorage.sol

x

- ├── clp
 - ├── alias.go
 - ├── client
 - ├── cli
 - ├── flags.go
 - ├── query.go
 - ├── tx.go
 - └── tx_test.go
 - └── rest
 - ├── query.go
 - ├── rest.go
 - └── tx.go
 - ├── clp_suite_test.go
 - ├── genesis.go
 - ├── genesis_test.go
 - ├── handler.go
 - ├── handler_test.go
 - ├── keeper
 - ├── calculations.go
 - ├── executors.go
 - ├── keeper.go
 - ├── keeper_test.go
 - ├── liquidityprovider.go
 - ├── params.go
 - ├── pool.go
 - ├── pureCalculation.go
 - ├── querier.go
 - ├── querier_test.go
 - └── table_test.go



```
|
|   | validatorWhiteList_test.go
|   | validatorWhitelist.go
|   | module.go
|   | test
|   |   | test_common.go
|   | types
|   |   | asset.go
|   |   | codec.go
|   |   | errors.go
|   |   | events.go
|   |   | expected_keepers.go
|   |   | genesis.go
|   |   | keys.go
|   |   | keys_test.go
|   |   | msgs.go
|   |   | msgs_test.go
|   |   | params.go
|   |   | querier.go
|   |   | types.go
| ethbridge
|   | alias.go
|   | client
|   |   | cli
|   |   |   | query.go
|   |   |   | tx.go
|   |   | module_client.go
|   |   | rest
|   |   |   | rest.go
|   | handler.go
|   | handler_test.go
|   | keeper
|   |   | cethReceiverAccount.go
|   |   | cethReceiverAccount_test.go
|   |   | keeper.go
|   |   | keeper_test.go
|   |   | peggyTokens.go
|   |   | querier.go
|   |   | querier_test.go
|   |   | test_common.go
|   | module.go
|   | test_common.go
|   | types
|   |   | claim.go
|   |   | claim_type.go
|   |   | codec.go
|   |   | errors.go
|   |   | ethereum.go
|   |   | events.go
|   |   | expected_keepers.go
|   |   | flags.go
|   |   | keys.go
|   |   | msgs.go
|   |   | querier.go
|   |   | test_common.go
| oracle
|   | alias.go
|   | genesis.go
|   | keeper
```




```
| |— adminAccount.go
| |— adminAccount_test.go
| |— keeper.go
| |— keeper_test.go
| |— test_common.go
| |— validatorWhiteList.go
| |— validatorWhiteList_test.go
|— module.go
|— types
|   |— claim.go
|   |— codec.go
|   |— errors.go
|   |— expected_keepers.go
|   |— genesis.go
|   |— keys.go
|   |— prophecy.go
|   |— status.go
```

Methodology

The audit has been performed by a mixed team of smart contract and full-stack auditors.

The following steps were performed:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
 - a. Race condition analysis
 - b. Under- / overflow issues
 - c. Key management vulnerabilities
 - d. Permissioning issues
 - e. Logic errors
4. Report preparation

The results were then discussed between the auditors in a consensus meeting and integrated into this joint report.

Functionality Overview

The submitted code implements the continuous liquidity pool functionality built into the Sifchain protocol. Liquidity pools can be created for asset pairs, allowing swaps between Sifchain's native token (Rowan) and Sifchain representations of external tokens, providing the basic DEX facility offered by the protocol.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria for each module, in the corresponding findings section.

Note, that high complexity or lower test coverage generally equate to a higher remaining risk. Test coverage is of particular importance, since certain bugs are more easily detected in unit testing than a security audit and vice versa.

Notes on Architecture / Functionality

Sifchain CLP

Apart from the code implementation security review, the audit team has performed a review of the currently supported functionality and architecture.

The Sifchain CLP architecture builds directly on Cosmos SDK (<https://cosmos.network/sdk>) and uses the Cosmos-native primitives and the same terminology. The Cosmos SDK version used in the audited code was `v0.39.2`.

Liquidity pools in Sifchain provide liquidity for pairings of external assets to Sifchain's native token.

IMPORTANT NOTICE: The audit does not evaluate the merits and validity of the economic model. It merely verifies that the implementation matches the specification.

Peggy Ethereum Bridge

Apart from the code implementation security review, the audit team has performed a review of the currently supported functionality and architecture. The following a number of informal observations:

- **Ethereum Transaction Relaying:**

The relayer currently does not relay data from transactions directly from Ethereum to the peg zone. Instead, it subscribes to events emitted through the smart contracts. This is an efficient way to deal with this but may lead to inconsistencies if events are missed.

- **Finality Considerations: (UPDATE: Resolved)**

There is also currently no evidence of a solution to the finality mismatch between the Ethereum and Cosmos chain. Tendermint consensus used by Cosmos provides instant finality through a byzantine voting protocol. Ethereum, on the other hand, provides probabilistic consensus, which means that blocks can become state and transactions may be rolled back retrospectively initially.

Peg zones usually act as a “finality gadget buffer” in this. Whilst it is clear that the finality gadget is intended for future versions, the current design does not lend itself to implement this without major refactoring.

UPDATE: The team has introduced 50-block delay before submitting events to Cosmos chain to address this issue

- **Supported Assets:**



Currently, ETH and ERC-20 tokens are supported. Whilst focusing on ERC-20 assets is a good choice, care must be taken with malicious implementations, copycat tokens, and compatible standards, such as ERC-777 tokens that may inject arbitrary code through callbacks.

Sifchain CLP Findings

Summary

The Sifchain CLP component was found to contain the issues summarize in the following table and detailed below:

No	Description	Severity	Status
1	x/clp/keeper/Calculations.go: Token decimal adjustment is always executed	Minor	Pending
2	Limitations of pool addressing convention	Minor	Pending
3	Gas limitations	Informational	-
4	Duplicate function implementations in calculations.go	Informational	-
5	Duplicate logic in calculations.go	Informational	-
6	Leftover TODO comments	Informational	-
7	Documentation refers to Thorchain	Informational	-

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	Medium	-

Detailed Findings

1. `x/clp/keeper/Calculations.go`: Token decimal adjustment is always executed

Severity: Minor

In functions `calcLiquidityFee()` and `calcSwapResult()`, there are checks to see if the number of decimals of external tokens need to be normalized. However, the actual adjustment code is executed in either case, due to the following incorrect if statement:

```
if adjustExternalToken {
    if toRowan {
        X = X.Mul(sdk.NewUintFromBigInt(normalizationFactor.RoundInt().BigInt()))
        x = x.Mul(sdk.NewUintFromBigInt(normalizationFactor.RoundInt().BigInt()))
    } else {
        Y = Y.Mul(sdk.NewUintFromBigInt(normalizationFactor.RoundInt().BigInt()))
    }
} else {
    if toRowan {
        X = X.Mul(sdk.NewUintFromBigInt(normalizationFactor.RoundInt().BigInt()))
        x = x.Mul(sdk.NewUintFromBigInt(normalizationFactor.RoundInt().BigInt()))
    } else {
        Y = Y.Mul(sdk.NewUintFromBigInt(normalizationFactor.RoundInt().BigInt()))
    }
}
```

Recommendation

Remove the else statement.

Status: Pending

2. Limitations of pool addressing convention

Severity: Minor

CLPs are identified using the symbol. According to the documentation the hash of the `pool-[firstTicker][secondTicker]` naming convention is used, even though only symbol key lookup can be evidenced in the code.

In either case, this addressing scheme means that the token symbol is used (as throughout Sifchain) to identify individual assets. This could lead to collisions with assets using the same token symbols.

Recommendation

Consider revising the pool addressing scheme or ensure that collisions are avoided through whitelisting.

Status: Pending

3. Gas Limitations

Severity: Informational

The implementation of the CLP seems to depend on Tendermint being configured to have an infinite block gas limit, which is the default. This is due to iterations over variable-sized data structures. In particular, this is the case for destroying a liquidity pool, in which case the code iterates through all liquidity providers to refund funds. If a limit is placed on the amount of gas that can be consumed in a block, the transactions may fail if the number of liquidity providers is too large.

Recommendation

Consider avoiding looping over variable-sized arrays or ensure that no gas limit is applied to the chain.

4. Duplicate function implementations in calculations.go

Severity: Informational

The following functions have identical duplicate implementations in both `calculations.go` and `pureCalculation.go`: `ValidateZero()`, `ReducePrecision()`, `IncreasePrecision()`, `GetMinLen()`

Recommendation

Consider removing the duplicate implementations from `calculations.go` and using the `pureCalculation.go` version instead.

5. Duplicate logic in calculations.go

Severity: Informational

In `calculations.go`, both the `SwapOne()` and `GetSwapFee()` functions share the following duplicate logic:

```
var X sdk.Uint
var Y sdk.Uint
toRowan := true
if to == types.GetSettlementAsset() {
    Y = pool.NativeAssetBalance
```




```
        X = pool.ExternalAssetBalance
    } else {
        X = pool.NativeAssetBalance
        Y = pool.ExternalAssetBalance
        toRowan = false
    }
```

Recommendation

As a best practice, consider grouping duplicate logic in its own function.

6. Leftover TODO comments

Severity: Informational

In `abci.go`, no begin- or end blocks are defined and the empty functions are still marked with the auto-generated TODO comments.

Recommendation

Consider removing the functions for code clarity.

7. Documentation refers to Thorchain

Severity: Informational

The CLP documentation refers to Thorchain instead of Sifchain throughout.

Recommendation

Consider updating the documentation.

Peggy Ethereum Bridge Findings

Solidity Smart Contracts

Summary of Findings

The Sifchain Peggy smart contracts were found to contain the issues summarize in the following table and detailed below:

No	Description	Severity	Status
1	Potential namespace collisions through token symbol	Major	Resolved
2	Potential for funds to be locked indefinitely in case the locked ERC20 tokens do not support returning values on transfer()	Major	Resolved
3	Potential for BridgeBank.lock() to always fail in case the it attempts to lock an ERC20 token that does not support returning values on transfer()	Minor	Resolved
4	Missing checks for maximum consensus threshold	Minor	Resolved
5	Malleable signatures	Minor	Resolved
6	Duplicate validators are accepted in validator set	Minor	Acknowledged / Partially Resolved
7	Ether transfers depending on gas stipend may not work if the recipient is a smart contract	Minor	Resolved
8	Compiler version	Informational	Resolved
9	Misleading behavior for recoverGas function	Informational	Resolved
10	Bridge bank can receive ETH directly	Informational	Resolved
11	Unnecessary modifier	Informational	Resolved



12	Unnecessary protection in view function	Informational	Resolved
13	Optimization on token deployment costs	Informational	Acknowledged / Partially Resolved
14	Consider using a mock address for ETH	Informational	-
15	Incorrect natspect comments	Informational	-
16	Node package inclusion in Solidity files breaks build system	Informational	-
17	Provide formal documentation on why the number 30 Ethereum blocks was chosen as safe enough finality for Sifchain.	Informational	-

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	Medium-high	-
Test Coverage	Medium	-

Detailed Findings

1. Potential namespace collisions through token symbol

Severity: Major

Tokens are identified by their symbol in both `EthereumBank.sol` and `CosmosBank.sol`. This will lead to namespace collisions and may potentially be exploited by copycat tokens.

Some tokens can also change their symbol (example: DAI → SAI).

Recommendation

Keep track of tokens in a way that avoids name clashes. For example, have a separate token registry that maps tokens to a unique ID.

Status: Resolved

The team has added checks to ensure a symbol can only be registered once.

2. Potential for funds to be locked indefinitely in case the locked ERC20 tokens do not support returning values on transfer()

Severity: Major

Function `unlockFunds()` in `EthereumBank.sol` assumes `transfer()` returns a value. If the locked token's `transfer()` function does not return a value, `EthereumBank.unlockFunds()` will never succeed and the funds will remain forever locked.

Recommendation

Consider using OpenZeppelin's `safeTransfer()` instead of `transfer()`.

Status: Resolved

3. Potential for BridgeBank.lock() to always fail in case the it attempts to lock an ERC20 token that does not support returning values on transfer()

Severity: Minor

Function `lock()` in `BridgeBank.sol` assumes `transferFrom()` returns a value. If the token's `transferFrom()` function does not return a value, `BridgeBank.lock()` will never succeed.

Recommendation

Consider using OpenZeppelin's `safeTransfer()` instead of `transfer()`.

Status: Resolved

4. Missing checks for maximum consensus threshold

Severity: Minor

`Oracle.sol` declares a consensus threshold as a `uint256`. Whilst this is meant to be a percentage no maximum is enforced. Numbers above 100 will lead to consensus not being achievable.

Recommendation

Enforce a check that limits the consensus threshold to a number between 1 and 100 in the constructor.

Status: Resolved

5. Malleable signatures

Severity: Minor

The `verify()` function in `Valset.sol`, the built-in `ecrecover()` function is used. This function still allows malleable signatures for backward compatibility reasons. Signatures that have an `s` value larger than

`0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0` are usually rejected for Ethereum addresses post EIP-2¹.

Recommendation

Consider rejecting signatures with `s` values in the upper ranges, even though it may not be a security issue in this case.

Status: Resolved

6. Duplicate validators are accepted in validator set

Severity: Minor

The `Valset.sol` contract does not check for duplicate validator additions. This may produce inconsistencies related to voting power and consensus. The `addValidator()` function also does not check if a validator already exists.

Recommendation

¹ <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2.md>

Enforce uniqueness in the `add` function, as well unique addresses in the constructor and `updateValset()` function. A cheap way to do this is to require addresses to be ordered in the array.

Status: Acknowledged / Partially Resolved

The team has resolved the issue by providing a unique key for each validator. the solution introduces a potential inconsistency in the validator count, since validator counts are updated even if an already existing validator is added (overwritten). However, the team prioritizes the benefit of allowing a validator's voting count to be updated in this way.

7. Ether transfers depending on gas stipend may not work if the recipient is a smart contract

Severity: Minor

The `transfer()` function is used to transfer ETH in `EthereumBank.sol` (line 158). With the Istanbul hard fork gas prices changed meaning the forwarded gas of 2300 may not be enough for smart contract recipients to perform their operations, leading to the call reverting.

Recommendation

It is now recommended to use `call.value()` for transferring ETH.

Status: Resolved

8. Compiler version

Severity: Informational

The code uses `pragma solidity ^0.5.0;` to define the compiler version to be used. Whilst we do not recommend using the latest compiler version due to the risk of undiscovered bugs, when using Solidity 0.5, the latest maintenance release (0.5.17) should be used since a number of bug fixes have been applied.

Recommendation

Lock the pragma to version 0.5.17

Status: Resolved

The team locked the pragma to version 0.5.16.

9. Misleading behavior for recoverGas function

Severity: Informational

Refunds are capped at the half of the transaction cost, so it only reduces gas cost if run within a bigger transaction to reduce the overall cost, otherwise there will be a net loss in executing the function, which is misleading from what the name implies.

Recommendation

Make it more clear the usage and functionality or consider renaming.

Status: Resolved

10. Bridge bank can receive ETH directly

Severity: Informational

BridgeBank.sol implements an operator fallback function for receiving ETH directly marked “for testing purposes”

Recommendation

Consider removing the leftover test code.

Status: Resolved

11. Unnecessary modifier

Severity: Informational

EthereumBank.sol implements the following modifier for avoiding nonce overflow:

```
modifier availableNonce() {  
    require(lockBurnNonce + 1 > lockBurnNonce, "No available  
nonces.");  
    _;  
}
```

However, new nonces are always generated using the `safeMath` library, which already protects the system from overflows. In any case, the system would stop functioning if the nonce space was exhausted (which is not a realistic possibility with 256-bit numbers).

Recommendation

Remove the modifier for efficiency.

Status: Resolved

12. Unnecessary protection in view function

Severity: Informational

The `checkBridgeProphecy()` function checks that the caller is an operator, even though it does not change the state and all information is public.

Recommendation

Remove the modifier for efficiency.

Status: Resolved

13. Optimization on token deployment costs

Severity: Informational

Since peggy Tokens are deployed by a transaction on-chain, using a proxy pattern might decrease the gas usage significantly, instead of deploying a full token contract every time.

Recommendation

Implement the pattern for increased efficiency.

Status: Acknowledged

The team acknowledges the cost of token deployment and has applied some gas optimizations to mitigate.

14. Consider using a mock address for ETH

Severity: Informational

In the contracts, for referencing a eth value transfer the token address is set to `0x0`, however this is already the default value mappings. While there's no security implication as is, a better option would be to choose a mock address for ETH, like `0xeeeee...` to avoid any possible confusion with uninitialized addresses.

Recommendation

Change the ETH token address

15. Incorrect natspect comments

Severity: Informational

The functions `burnFunds()` and `lockFunds()` both have incorrect natspec comments

Recommendation

Adjust the comments to the appropriate behavior.

16. Node package inclusion in Solidity files breaks build system

Severity: Informational

The smart contract course files include OpenZeppelin dependencies imported as npm packages with relative path names. For example:

```
import "../../node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol";
```

This breaks the functionality of most up to date development frameworks, including Truffle used in the project, leading to the project not currently building on a clean installation.

Recommendation

Import npm dependencies directly, since they are discoverable from Truffle:

```
import "openzeppelin-solidity/contracts/math/SafeMath.sol";
```

17. Provide formal documentation on why the number 30 Ethereum blocks was chosen as safe enough finality for Sifchain.

Severity: Informational

Provide more explicit documentation of the security assumptions that made Sifchain arrive at the fact that 30 blocks is safe finality for the purposes of the Sifchain application.

Go Components

Summary of Findings

The Sifchain Peggy Go components were found to contain were found to contain the issues summarize in the following table and detailed below:

No	Description	Severity	Status
15	Outdated indirect dependency with multiple known security vulnerabilities	Minor	Pending
16	Relayer quits silently on failure to connect to subscribe to Ethereum events	Minor	Pending
17	Missing default case in switch statement	Informational	-
18	Empty code blocks	Informational	-

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	N/A	<p>No automated unit tests were supplied with the codebase to be audited. Therefore, no reliable measure of test coverage can be taken.</p> <p>However, there is evidence of extensive manual testing in the form of test code and scripts.</p>

Detailed Findings

18. Outdated indirect dependency with multiple known security vulnerabilities

Severity: Minor

The codebase uses an outdated version of the `ecdt` library (versions 3.3.13). This version includes a number of known vulnerabilities, including TLS authentication issues, TCP proxy discovery issues and missing password length verifications.

Recommendation

Update the dependency.

Status: Pending

19. Relay quits silently on failure to connect to subscribe to Ethereum events

Severity: Minor

The launch code of relay just terminates the process if the Ethereum connection is not available:

```
client, err := SetupWebsocketEthClient(sub.EthProvider)
if err != nil {
    sub.Logger.Error(err.Error())
    os.Exit(1)
}
```

This may cause issues in deployment scenarios, since it does not permit monitoring an automatic relaunching easily.

Relevant code sections:

- `cmd/ebrelayer/main.go` 163, 164
- `cmd/ebrelayer/relayer/ethereum.go` 119

Recommendation

Fail gracefully reporting an error code similar that can be used in an infrastructure management scenario.

Status: Pending

20. Missing default case in switch statement

Severity: Informational

The switch statement for processing Ethereum event data in `cmd/ebrelayer/relayer/ethereum.go:158-168` is lacking a default case. It is always recommended to add default cases for clarity and dealing with unexpected data.

Recommendation

Add a default case.

21. Empty code blocks

Severity: Informational

The codebase contains several places where unimplemented functions are marked as TODO. Whilst this is common practise during development, this should be avoided in production code.

Relevant code sections:

- `x/ethbridge/module.go` 108, 145
- `x/oracle/module.go` 33, 48, 89, 124

Recommendation

Clean up TODO comments.