Maya Protocol - Thornode

Cosmos Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 4th, 2022 - October 10th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/08/2022	John Saigle
0.2	Document Updates	10/10/2022	Emiliano Carmona
0.3	Draft Review	10/13/2022	Gabi Urrutia
1.0	Remediation Plan	01/25/2023	John Saigle
1.1	Remediation Plan Updates	01/25/2023	Gokberk Gulgun
1.2	Remediation Plan Review	01/25/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
John Saigle	Halborn	John.Saigle@halborn.com
Emiliano Carmona	Halborn	Emiliano.Carmona@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Maya Protocol engaged Halborn to conduct a security audit on their protocol beginning on September 4th, 2022 and ending on October 10th, 2022 . The security assessment was scoped to the Cosmos modules provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned two full-time security engineers to audit the security of the project. The security engineers are a blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that Maya modules are functioning as intended.
- Identify potential security issues and report them to the Maya team.

In summary, Halborn identified some security risks in the project. The Maya Protocol team discussed the findings with Halborn and addressed all major issues there were discovered during the engagement.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the custom modules. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of structures and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Static Analysis of security for scoped repository, and imported functions. (e.g., staticcheck, gosec, unconvert, codeql, ineffassign and semgrep)
- Manual Assessment for discovering security vulnerabilities on codebase.
- Ensuring correctness of the codebase.
- Dynamic Analysis on files and modules related to the Maya Protocol.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. IN-SCOPE TREE & COMMIT

The review was scoped to Maya's THORNode project repository at the following URL:

- https://gitlab.com/mayachain/thornode

The review was conducted on the following commit hash:

- 9deb084ff5720e1e1c9d28c95d0c2c94687ca925

2. REMEDIATION PRs & COMMITS:

- https://gitlab.com/mayachain/thornode/-/merge_requests/31
- https://gitlab.com/mayachain/thornode/-/merge_requests/32
- https://gitlab.com/mayachain/thornode/-/merge_requests/27
- https://gitlab.com/mayachain/thornode/-/merge_requests/33
- https://gitlab.com/mayachain/thornode/-/merge_requests/41
- https://gitlab.com/mayachain/thornode/-/merge_requests/31
- https://gitlab.com/mayachain/thornode/-/merge_requests/23
- https://gitlab.com/mayachain/thornode/-/merge_requests/37
- https://gitlab.com/mayachain/thornode/-/merge_requests/38
- https://gitlab.com/mayachain/thornode/-/merge_requests/39

Commit IDS :

- e949e07f1d3f875ad6decdd96daf416baa49dc1f

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	0	12	4

LIKELIHOOD

IMPACT

🟨	🟨	🟥 (HAL-01)	🟥	🟪
🟩 (HAL-03) (HAL-04) (HAL-07)	🟨	🟨	🟥	🟥 (HAL-02)
🟩 (HAL-05) (HAL-06) (HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-14)	🟩	🟨	🟨	🟥
⬜	🟩 (HAL-12) (HAL-13)	🟩	🟨	🟨
⬜ (HAL-15) (HAL-16) (HAL-17) (HAL-18)	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - SLASHING LOGIC TERMINATES EARLY WHICH MAY PREVENT MALICIOUS NODES FROM BEING SLASHED	High	SOLVED - 25/01/23
HAL-02 - INCORRECT DISTRIBUTION OF FUNDS DUE TO PRECISION LOSS	High	RISK ACCEPTED
HAL-03 - LIQUIDITY PROVIDERS MAY LOSE FUNDS DURING A RAGNAROK DUE TO ERROR IN REFUND LOGIC	Low	SOLVED - 25/01/23
HAL-04 - SYNTHETIC ASSETS CAN LOSE COLLATERALIZATION PROTECTION DUE TO AN INTEGER OVERFLOW	Low	SOLVED - 25/01/23
HAL-05 - PER-BLOCK REFUNDS ARE LARGER THAN INTENDED DURING RAGNAROK	Low	SOLVED - 25/01/23
HAL-06 - OVERLY PERMISSIVE CORS POLICY	Low	RISK ACCEPTED
HAL-07 - FUNCTION SETMIMIR FAILS SILENTLY WHEN DISABLED	Low	SOLVED - 25/01/23
HAL-08 - NO COMPLEXITY REQUIREMENTS FOR BIFROST PASSPHRASE	Low	SOLVED - 25/01/23
HAL-09 - USE OF INSECURE HASH FUNCTION MD5	Low	RISK ACCEPTED
HAL-10 - LACK OF KEY-DERIVATION FUNCTION USED FOR AES ENCRYPTION KEY	Low	RISK ACCEPTED
HAL-11 - LOGIC PROBLEMS CAN OCCUR WHEN CONVERTING MIMIR VALUES BETWEEN SIGNED AND UNSIGNED INTEGERS	Low	RISK ACCEPTED
HAL-12 - UNHANDLED ERRORS	Low	SOLVED - 25/01/23
HAL-13 - USE OF OUTDATED COSMOSSDK VERSION	Low	SOLVED - 25/01/23
HAL-14 - VULNERABLE THIRD PARTY PACKAGES	Informational	SOLVED - 25/01/23
HAL-15 - USE OF DEPRECATED FUNCTIONS	Informational	SOLVED - 25/01/23

HAL-16 - PANICS USED AS ERROR-HANDLING	Informational	ACKNOWLEDGED
HAL-17 - OPEN TODOS IN CODEBASE	Informational	ACKNOWLEDGED
HAL-18 - SPELLING MISTAKES IN THE CODEBASE	Informational	SOLVED - 25/01/23

FINDINGS & TECH DETAILS

3.1 (HAL-01) SLASHING LOGIC TERMINATES EARLY WHICH MAY PREVENT MALICIOUS NODES FROM BEING SLASH - HIGH

Description:

Slashing is intended to provide an economic incentive for validators to behave honestly by removing their bond if they are found to act in malicious ways.

The slashing logic is implemented below by looping over all the coins supported by the vault, and then entering a second loop inside the first loop that iterates through all the accounts for a given Vault. All accounts who have bonded to this vault should be slashed when malicious behavior occurs.

Each iteration of the loop can change the state of the chain by removing LP units (Line 513).
However, the loop has several locations in which a return statement is issued when an error is encountered. These return statements will cause the entire function to return an error and exit both for-loops.

This could result in situations where only some coins or accounts are slashed during the initial iterations of the loop, and other accounts will not be slashed due to the loops exiting early. This could result in a scenario where nodes can behave maliciously without consequences.

Code Location:

x/thorchain/manager_slasher_current.go function SlashVaultToLP Lines 427-521

Listing 1: Vulnerable logic with early returns

```
// Slash for each coins.
    for _, coin := range coins {
        ...
        // Slash each node account in the vault.
        for _, member := range membership {
            na, err := s.keeper.GetNodeAccountByPubKey(ctx, member)
            if err != nil {
                ctx.Logger().Error("fail to get node account for slash", "pk", member, "error", err)
                continue
            }
            naBond, err := mgr.Keeper().CalcNodeLiquidityBond(ctx, na)
            if err != nil {
                return ErrInternal(err, "fail to get node account bond")
            }
            if naBond.IsZero() {
                ctx.Logger().Info("validator's bond is zero, can't be slashed", "node address", na.NodeAddress.String())
                continue
            }
            slashAmountRune := common.GetSafeShare(naBond, totalBondInVault, totalSlashAmountInRune)
            if slashAmountRune.GT(naBond) {
                ctx.Logger().Info("slash amount is larger than bond", "slash amount", slashAmountRune, "bond", naBond)
                slashAmountRune = naBond
            }

            // Slash correspoding lpunits proportionally to LP.
            bp, err := mgr.Keeper().GetBondProviders(ctx, na.NodeAddress)
            if err != nil {
                return ErrInternal(err, "fail to get node bond providers")
            }
            for _, b := range bp.Providers {
                lps, err := mgr.Keeper().GetLiquidityProviderByAssets(ctx, LiquidityPools, common.Address(b.BondAddress.String()))
                if err != nil {
```

```go
                return ErrInternal(err, fmt.Sprintf("fail to get lps for bond provider: %s", b.BondAddress))
            }
            for _, lp := range lps {
            pool, err := mgr.Keeper().GetPool(ctx, lp.Asset)
                if err != nil {
                    return ErrInternal(err, "fail to get pool")
                }

                if pool.IsAvailable() {
                    // Calculate corresponding lp units to take away.
                    lpBondedRune := common.GetSafeShare(lp.Units, pool.LPUnits, pool.BalanceRune)
                    slashRuneForLP := common.GetSafeShare(lpBondedRune, naBond, slashAmountRune)
                    slashLPUnits := common.GetSafeShare(slashRuneForLP, pool.BalanceRune, pool.LPUnits)

                    // Take away corresponding lp units due to slash.
                    lp.Units = lp.Units.Sub(slashLPUnits)
                    mgr.Keeper().SetLiquidityProvider(ctx, lp)
                }
            }
        }

        naBond, err = mgr.Keeper().CalcNodeLiquidityBond(ctx, na)
        if err != nil {
            return ErrInternal(err, "fail to get node account bond")
        }
```

Example 1 x/thorchain/manager_yggdrasil_current.go
Lines 436-443

Listing 2: Slash function is called but execution continues if there is an error

```go
436      if err := ymgr.slash(ctx, slasher, mgr, na.PubKeySet.Secp256k1, v); err != nil {
437          ctx.Logger().Error("fail to slash node account", "key", na.PubKeySet.Secp256k1, "error", err)
438          continue
439      }
440       // assume slash finished successfully, delete the yggdrasil vault
441         if err := ymgr.keeper.DeleteVault(ctx, na.PubKeySet.Secp256k1); err != nil {
442             ctx.Logger().Error("fail to delete yggdrasil vault", "key", na.PubKeySet.Secp256k1, "error", err)
443         }
```

x/thorchain/manager_yggdrasil_current.go
Lines 449-457

Listing 3: Contents of the slash function. Error returned from SlashVaultToLP is not checked and gets overwritten by the following function call

```go
449 func (ymgr YggMgrV79) slash(ctx cosmos.Context, slasher Slasher, mgr Manager, pk common.PubKey, ygg Vault) error {
450     ctx.Logger().Info(fmt.Sprintf("slash, node account %s churned out , but fail to return yggdrasil fund", pk.String()), "coins", ygg.Coins.String())
451     err := slasher.SlashVaultToLP(ctx, pk, ygg.Coins, mgr)
452     ygg.SubFunds(ygg.Coins)
453     if err := ymgr.keeper.SetVault(ctx, ygg); err != nil {
454         return fmt.Errorf("fail to save yggdrasil vault: %w", err)
455     }
456     return err
457 }
```

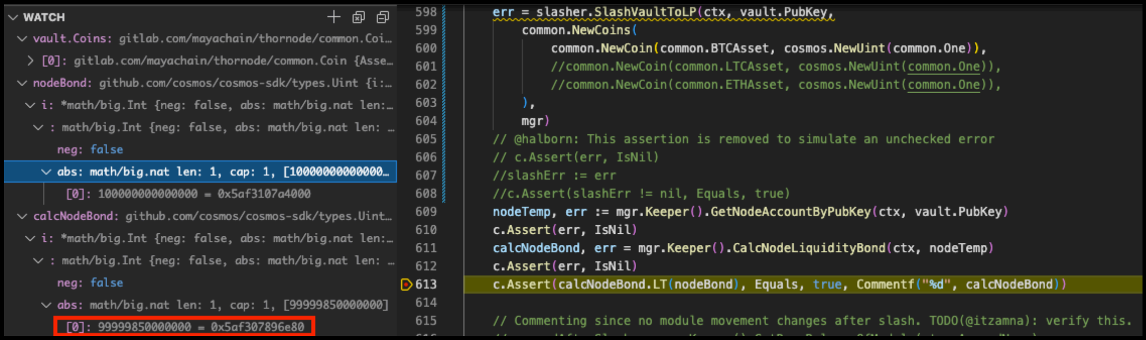
Example 2 x/thorchain/handler_observed_txout.go
Lines 176-199

Listing 4: Error returned from SlashVaultToLP is not called

```go
if memo.IsEmpty() || memo.IsInbound() {
    vault, err := h.mgr.Keeper().GetVault(ctx, tx.ObservedPubKey)
        if err != nil {
            ctx.Logger().Error("fail to get vault", "error", err)
                continue
        }
        toSlash := tx.Tx.Coins.Adds(tx.Tx.Gas.ToCoins())

        slashCtx := ctx.WithContext(context.WithValue(ctx.Context
 ↳ (), constants.CtxMetricLabels, []metrics.Label{
                telemetry.NewLabel("reason", "sent_extra_funds"),
                telemetry.NewLabel("chain", string(tx.Tx.Chain)),
                }))

        if err := h.mgr.Slasher().SlashVaultToLP(slashCtx, tx.
 ↳ ObservedPubKey, toSlash, h.mgr); err != nil {
            ctx.Logger().Error("fail to slash account for sending
 ↳ extra fund", "error", err)
        }
        vault.SubFunds(toSlash)
        if err := h.mgr.Keeper().SetVault(ctx, vault); err != nil
 ↳ {
            ctx.Logger().Error("fail to save vault", "error", err)
        }

        continue
}
```

Proof-of-concept:

In this scenario, it is simulated a single validator who has staked tokens.

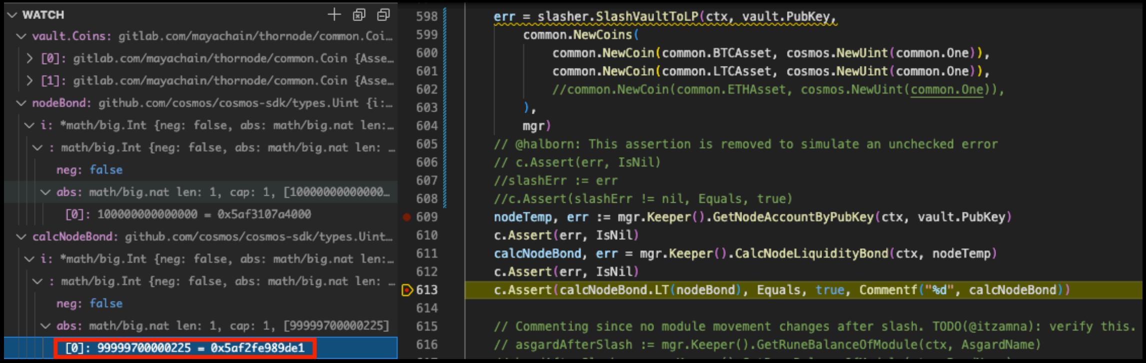


```
WATCH
vault.Coins: gitlab.com/mayachain/thornode/common.Coi…
  [0]: gitlab.com/mayachain/thornode/common.Coin {Asse…
nodeBond: github.com/cosmos/cosmos-sdk/types.Uint {i:…
  i: *math/big.Int {neg: false, abs: math/big.nat len:…
    : math/big.Int {neg: false, abs: math/big.nat len: …
      neg: false
      abs: math/big.nat len: 1, cap: 1, [10000000000000…
        [0]: 100000000000000 = 0x5af3107a4000
calcNodeBond: github.com/cosmos/cosmos-sdk/types.Uint…
  i: *math/big.Int {neg: false, abs: math/big.nat len:…
    : math/big.Int {neg: false, abs: math/big.nat len: …
      neg: false
      abs: math/big.nat len: 1, cap: 1, [99999850000000]
        [0]: 99999850000000 = 0x5af307896e80
```

```go
	err = slasher.SlashVaultToLP(ctx, vault.PubKey,
		common.NewCoins(
			common.NewCoin(common.BTCAsset, cosmos.NewUint(common.One)),
			//common.NewCoin(common.LTCAsset, cosmos.NewUint(common.One)),
			//common.NewCoin(common.ETHAsset, cosmos.NewUint(common.One)),
		),
		mgr)
	// @halborn: This assertion is removed to simulate an unchecked error
	// c.Assert(err, IsNil)
	//slashErr := err
	//c.Assert(slashErr != nil, Equals, true)
	nodeTemp, err := mgr.Keeper().GetNodeAccountByPubKey(ctx, vault.PubKey)
	c.Assert(err, IsNil)
	calcNodeBond, err = mgr.Keeper().CalcNodeLiquidityBond(ctx, nodeTemp)
	c.Assert(err, IsNil)
	c.Assert(calcNodeBond.LT(nodeBond), Equals, true, Commentf("%d", calcNodeBond))

	// Commenting since no module movement changes after slash. TODO(@itzamna): verify this.
```

Figure 1: Node with one token is slashed

Here is the slashing logic demonstrated when a validator holds a single asset BTC. In this case, their original bond was 100000000000000 and after being slashed their bond is 99999850000000.

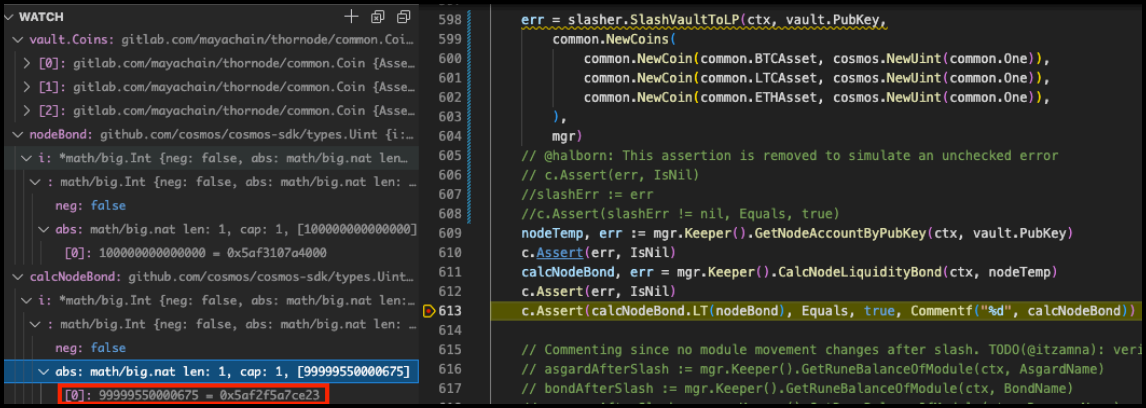


```
WATCH
vault.Coins: gitlab.com/mayachain/thornode/common.Coi…
  [0]: gitlab.com/mayachain/thornode/common.Coin {Asse…
  [1]: gitlab.com/mayachain/thornode/common.Coin {Asse…
nodeBond: github.com/cosmos/cosmos-sdk/types.Uint {i:…
  i: *math/big.Int {neg: false, abs: math/big.nat len:…
    : math/big.Int {neg: false, abs: math/big.nat len: …
      neg: false
      abs: math/big.nat len: 1, cap: 1, [10000000000000…
        [0]: 100000000000000 = 0x5af3107a4000
calcNodeBond: github.com/cosmos/cosmos-sdk/types.Uint…
  i: *math/big.Int {neg: false, abs: math/big.nat len:…
    : math/big.Int {neg: false, abs: math/big.nat len: …
      neg: false
      abs: math/big.nat len: 1, cap: 1, [99999700000225]
        [0]: 99999700000225 = 0x5af2fe989de1
```

```go
	err = slasher.SlashVaultToLP(ctx, vault.PubKey,
		common.NewCoins(
			common.NewCoin(common.BTCAsset, cosmos.NewUint(common.One)),
			common.NewCoin(common.LTCAsset, cosmos.NewUint(common.One)),
			//common.NewCoin(common.ETHAsset, cosmos.NewUint(common.One)),
		),
		mgr)
	// @halborn: This assertion is removed to simulate an unchecked error
	// c.Assert(err, IsNil)
	//slashErr := err
	//c.Assert(slashErr != nil, Equals, true)
	nodeTemp, err := mgr.Keeper().GetNodeAccountByPubKey(ctx, vault.PubKey)
	c.Assert(err, IsNil)
	calcNodeBond, err = mgr.Keeper().CalcNodeLiquidityBond(ctx, nodeTemp)
	c.Assert(err, IsNil)
	c.Assert(calcNodeBond.LT(nodeBond), Equals, true, Commentf("%d", calcNodeBond))

	// Commenting since no module movement changes after slash. TODO(@itzamna): verify this.
	// asgardAfterSlash := mgr.Keeper().GetRuneBalanceOfModule(ctx, AsgardName)
```

Figure 2: Node with two tokens is slashed

Here is the slashing logic when a validator holds two tokens, BTC and LTC. The bond is reduced to 99999700000225 because they are slashed based on the two separate tokens.

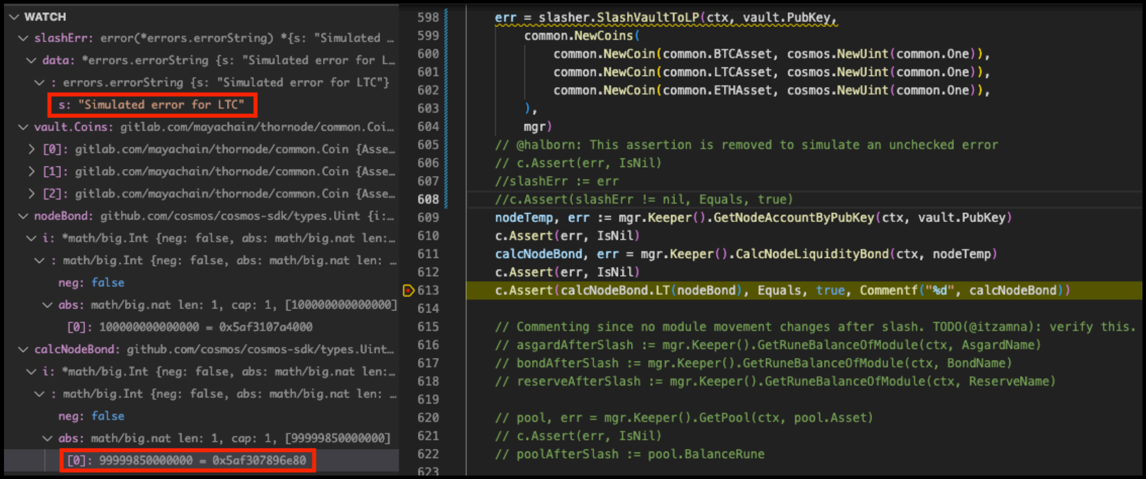


```
WATCH
vault.Coins: gitlab.com/mayachain/thornode/common.Coi…
  [0]: gitlab.com/mayachain/thornode/common.Coin {Asse…
  [1]: gitlab.com/mayachain/thornode/common.Coin {Asse…
  [2]: gitlab.com/mayachain/thornode/common.Coin {Asse…
nodeBond: github.com/cosmos/cosmos-sdk/types.Uint {i:…
  i: *math/big.Int {neg: false, abs: math/big.nat len…
    : math/big.Int {neg: false, abs: math/big.nat len: …
      neg: false
      abs: math/big.nat len: 1, cap: 1, [100000000000000]
        [0]: 100000000000000 = 0x5af3107a4000
calcNodeBond: github.com/cosmos/cosmos-sdk/types.Uint…
  i: *math/big.Int {neg: false, abs: math/big.nat len:…
    : math/big.Int {neg: false, abs: math/big.nat len: …
      neg: false
      abs: math/big.nat len: 1, cap: 1, [99999550000675]
        [0]: 99999550000675 = 0x5af2f5a7ce23
```

```go
	err = slasher.SlashVaultToLP(ctx, vault.PubKey,
		common.NewCoins(
			common.NewCoin(common.BTCAsset, cosmos.NewUint(common.One)),
			common.NewCoin(common.LTCAsset, cosmos.NewUint(common.One)),
			common.NewCoin(common.ETHAsset, cosmos.NewUint(common.One)),
		),
		mgr)
	// @halborn: This assertion is removed to simulate an unchecked error
	// c.Assert(err, IsNil)
	//slashErr := err
	//c.Assert(slashErr != nil, Equals, true)
	nodeTemp, err := mgr.Keeper().GetNodeAccountByPubKey(ctx, vault.PubKey)
	c.Assert(err, IsNil)
	calcNodeBond, err = mgr.Keeper().CalcNodeLiquidityBond(ctx, nodeTemp)
	c.Assert(err, IsNil)
	c.Assert(calcNodeBond.LT(nodeBond), Equals, true, Commentf("%d", calcNodeBond))

	// Commenting since no module movement changes after slash. TODO(@itzamna): veri
	// asgardAfterSlash := mgr.Keeper().GetRuneBalanceOfModule(ctx, AsgardName)
	// bondAfterSlash := mgr.Keeper().GetRuneBalanceOfModule(ctx, BondName)
```

Here is the slashing logic when a validator holds three tokens. The bond is once again even lower to 99999550000675 as the value of all three tokens is slashed.

In the proof-of-concept below, we show what happens when an error occurs when processing the LTC token.



```
WATCH
  slashErr: error(*errors.errorString) *{s: "Simulated …
    data: *errors.errorString {s: "Simulated error for L…
      : errors.errorString {s: "Simulated error for LTC"}
        s: "Simulated error for LTC"
  vault.Coins: gitlab.com/mayachain/thornode/common.Coi…
    > [0]: gitlab.com/mayachain/thornode/common.Coin {Asse…
    > [1]: gitlab.com/mayachain/thornode/common.Coin {Asse…
    > [2]: gitlab.com/mayachain/thornode/common.Coin {Asse…
  nodeBond: github.com/cosmos/cosmos-sdk/types.Uint {i:…
    i: *math/big.Int {neg: false, abs: math/big.nat len:…
      : math/big.Int {neg: false, abs: math/big.nat len: …
        neg: false
        abs: math/big.nat len: 1, cap: 1, [100000000000000]
          [0]: 100000000000000 = 0x5af3107a4000
  calcNodeBond: github.com/cosmos/cosmos-sdk/types.Uint…
    i: *math/big.Int {neg: false, abs: math/big.nat len:…
      : math/big.Int {neg: false, abs: math/big.nat len: …
        neg: false
        abs: math/big.nat len: 1, cap: 1, [99999850000000]
          [0]: 99999850000000 = 0x5af307896e80
```

```go
	err = slasher.SlashVaultToLP(ctx, vault.PubKey,
		common.NewCoins(
			common.NewCoin(common.BTCAsset, cosmos.NewUint(common.One)),
			common.NewCoin(common.LTCAsset, cosmos.NewUint(common.One)),
			common.NewCoin(common.ETHAsset, cosmos.NewUint(common.One)),
		),
		mgr)
	// @halborn: This assertion is removed to simulate an unchecked error
	// c.Assert(err, IsNil)
	//slashErr := err
	//c.Assert(slashErr != nil, Equals, true)
	nodeTemp, err := mgr.Keeper().GetNodeAccountByPubKey(ctx, vault.PubKey)
	c.Assert(err, IsNil)
	calcNodeBond, err = mgr.Keeper().CalcNodeLiquidityBond(ctx, nodeTemp)
	c.Assert(err, IsNil)
	c.Assert(calcNodeBond.LT(nodeBond), Equals, true, Commentf("%d", calcNodeBond))

	// Commenting since no module movement changes after slash. TODO(@itzamna): verify this.
	// asgardAfterSlash := mgr.Keeper().GetRuneBalanceOfModule(ctx, AsgardName)
	// bondAfterSlash := mgr.Keeper().GetRuneBalanceOfModule(ctx, BondName)
	// reserveAfterSlash := mgr.Keeper().GetRuneBalanceOfModule(ctx, ReserveName)

	// pool, err = mgr.Keeper().GetPool(ctx, pool.Asset)
	// c.Assert(err, IsNil)
	// poolAfterSlash := pool.BalanceRune
```

Figure 3: Slashing when an error occurs -- only BTC is slashed

In this case, the logic loops over the list of tokens starting with BTC, then LTC, and finally to ETH. BTC is slashed as expected. An error is returned when processing LTC. As a result, LTC is not slashed. Because the function returns early, ETH is also not slashed.

In this state, when a node is slashed, the following occurs:
- BTC can be successfully slashed
- LTC will return an error
- ETH cannot be slashed and is never processed.

The node's bond becomes 99999850000000 again in this case because only BTC is slashed.

All nodes are processed one after the other in the loop within the slashing function. Therefore, any node holding only ETH will be able to perform malicious actions without any consequences. Bonders using BTC will be punished. Bonders holding LTC will run into unexpected behavior because they will encounter errors and may also be able to behave maliciously.

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Ensure that when errors occur in the slashing logic that they are handled properly by calling code. If a slashing event occurs for a set of Tokens or Nodes, all relevant members should be slashed or none of them should be. This must be done to maintain the health of the network.

Remediation Plan:

SOLVED: The Maya Protocol team addressed this issue in a merge request which can be viewed at this link.

3.2 (HAL-02) INCORRECT DISTRIBUTION OF FUNDS DUE TO PRECISION LOSS - HIGH

Description:

The QuoUint64 function in the Cosmos SDK uses integer division. The function returns only the quotient, and the remainder is discarded. Therefore, precision loss can occur. For example: **12345 / 100 = 123** and the last two digits disappear.

Note that there is a risk of a further error here, as the very next line multiplies the result of the division using the same value it was divided by. Due to the quotient math above, multiplication does not undo the division. The order of operations is important.

E.g. multiply before divide
- 12345 x 100 = 1234500
- 1234500 / 100 = **12345**

divide before multiply
- 12345 / 100 = 123
- 123 x 100 = **12300**

In this case, the token.Amount is divided by minMultiple. If minMultiple does not divide evenly into token.Amount then some of the amount will be lost. This reduces the total amount that is divided into shares, so each participant gets less than they deserve.

Code Location:

Listing 5: POC output (Lines 2,3)

```
 1        if !mayaBalance.IsZero() {
 2          millionToken := token.Amount.QuoUint64((uint64)(minMultiple))
 3          tokenAmt := sdk.NewUint((uint64)(minMultiple)).Mul(millionToken)
 4          mayaAmt := common.GetSafeShare(sdk.NewUint(mayaBalance.Uint64()), totalAmountOfMaya, tokenAmt)
 5          mayaCoins := common.NewCoins(common.NewCoin(token.Asset, mayaAmt))
 6
 7          err := k.SendFromModuleToAccount(ctx, MayaFund, acc.GetAddress(), mayaCoins)
 8          if err != nil {
 9            ctx.Logger().Error("fail to send RUNE on MayaFund", "error", err)
10          }
11        }
12      }
13    }
```

Proof of concept:

Listing 6: POC output

```

=== RUN   Test
Rune balance of module set to:          1234567891
Amount to be distributed:               1234567891
Actual balance distributed:             1200000000

```

The above output was generated with the following script

Listing 7: POC script

```
package keeperv1

```

```go
import (
	"fmt"
	"testing"

	. "gopkg.in/check.v1"

	"gitlab.com/mayachain/thornode/common"
	"gitlab.com/mayachain/thornode/common/cosmos"
	"gitlab.com/mayachain/thornode/constants"
)

func Test(t *testing.T) { TestingT(t) }

type KeeperNetworkFeeSuitePrecisionTest struct{}

var _ = Suite(&KeeperNetworkFeeSuitePrecisionTest{})

func (KeeperNetworkFeeSuitePrecisionTest) TestNetworkFee(c *C) {
	ctx, k := setupKeeperForTest(c)
	networkFee := NewNetworkFee(common.BNBChain, 1, 37500)
	c.Check(k.SaveNetworkFee(ctx, common.BNBChain, networkFee),
↳ IsNil)

	networkFee1 := NewNetworkFee(common.BNBChain, 0, 37500)
	c.Check(k.SaveNetworkFee(ctx, common.BNBChain, networkFee1),
↳ NotNil)

	networkFee2, err := k.GetNetworkFee(ctx, common.ETHChain)
	c.Check(err, IsNil)
	c.Check(networkFee2.Valid(), NotNil)
	c.Check(k.GetNetworkFeeIterator(ctx), NotNil)
	networkFee3, err := k.GetNetworkFee(ctx, common.BNBChain)
	c.Check(err, IsNil)
	c.Check(networkFee3.Valid(), IsNil)
}

func (KeeperNetworkFeeSuitePrecisionTest) TestDistributeMayaFund(c
↳ *C) {
	ctx, k := setupKeeperForTest(c)

	// Mint MayaFund
	coin := common.NewCoin(common.RuneNative, cosmos.NewUint
↳ (1234567891))
	err := k.MintToModule(ctx, ModuleName, coin)
```

```go
    c.Assert(err, IsNil)
    err = k.SendFromModuleToModule(ctx, ModuleName, MayaFund,
↳ common.NewCoins(coin))
    fmt.Print("Rune balance of module set to:       ", coin.Amount
↳ , "\n")

    addr1 := GetRandomBech32Addr()

    acc1 := k.accountKeeper.NewAccountWithAddress(ctx, addr1)

    // In/Out values
    amtAcc1 := (uint64)(1234567891)
    fmt.Print("Amount to be distributed:        ", amtAcc1, "\n")
    FundAccountMayaToken(c, ctx, k, acc1.GetAddress(), amtAcc1)

    v := GetCurrentVersion()
    constantAccessor := constants.GetConstantValues(v)
    k.DistributeMayaFund(ctx, constantAccessor)

    // Get balances
    balAcc1 := k.GetBalance(ctx, acc1.GetAddress())

    for _, coin := range balAcc1 {
        if coin.GetDenom() == common.RuneNative.Native() {
            //c.Assert(coin.Amount.Equal(cosmos.NewInt((int64)(
↳ amtAcc1))), Equals, true)
            fmt.Print("Actual balance distributed:      ", coin.
↳ Amount, "\n")
        }
    }

}

```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Short-term, evaluate whether the division operation is needed in this case. The line immediately following the division appears to multiply it by the same value. Add comments to the code explaining the need for the mathematical operations and what they are trying to achieve. This will clarify the code and reveal potential mistakes.

Longer-term, consider interpreting the values as `Dec` rather than `Int` to achieve a higher level of precision. The rational numbers in Go's `math/big` are another option.

Remediation Plan:

RISK ACCEPTED: The Maya Protocol team has stated that this loss of precision is desired and expected by design. A code comment was added in the codebase to clarify this decision.

3.3 (HAL-03) LIQUIDITY PROVIDERS MAY LOSE FUNDS DURING A RAGNAROK DUE TO ERROR IN REFUND LOGIC - LOW

Description:

When a chain becomes unhealthy, nodes can choose to refund all liquidity providers. This is called a Ragnarok.
Due to a lack of error handling, a refund transaction may fail silently. The calling code will then continue in the Ragnarok process, which includes slashing the validators in order to delete the liquidity units. This has the result of deleting liquidity units without transferring the corresponding funds. This could cause a loss of funds.

Code Location:

x/thorchain/manager_txout_current.go

Listing 8: TryAddTxOutItem has many failure cases

```go
func (tos *TxOutStorageV88) TryAddTxOutItem(ctx cosmos.Context,
↳ mgr Manager, toi TxOutItem) (bool, error) {
    outputs, err := tos.prepareTxOutItem(ctx, toi)
    if err != nil {
        return false, fmt.Errorf("fail to prepare outbound tx: %w"
↳ , err)
    }
    if len(outputs) == 0 {
        return false, ErrNotEnoughToPayFee
    }

    // blacklist binance exchange as an outbound destination. This
↳  is because
    // the format of THORChain memos are NOT compatible with the
↳ memo
    // requirements of binance inbound transactions.
    blacklist := []string{
        "bnb136ns6lfw4zs5hg4n85vdthaad7hq5m4gtkgf23", // binance
↳ CEX address
```

```go
    }
    for _, b := range blacklist {
        if toi.ToAddress.Equals(common.Address(b)) {
            return false, fmt.Errorf("non-supported outbound address")
        }
    }

    // calculate the single block height to send all of these txout items,
    // using the summed amount
    outboundHeight := common.BlockHeight(ctx)
    if !toi.Chain.IsTHORChain() && !toi.InHash.IsEmpty() && !toi.InHash.Equals(common.BlankTxID) {
        toi.Memo = outputs[0].Memo
        targetHeight, err := tos.calcTxOutHeight(ctx, mgr.GetVersion(), toi)
        if err != nil {
            ctx.Logger().Error("failed to calc target block height for txout item", "error", err)
        }
        if targetHeight > outboundHeight {
            outboundHeight = targetHeight
        }
        voter, err := tos.keeper.GetObservedTxInVoter(ctx, toi.InHash)
        if err != nil {
            ctx.Logger().Error("fail to get observe tx in voter", "error", err)
            return false, fmt.Errorf("fail to get observe tx in voter,err:%w", err)
        }

        // When the inbound transaction already has an outbound , the make sure the outbound will be scheduled on the same block
        if voter.OutboundHeight > 0 {
            outboundHeight = voter.OutboundHeight
        } else {
            voter.OutboundHeight = outboundHeight
            tos.keeper.SetObservedTxInVoter(ctx, voter)
        }
    }

    // add tx to block out
    for _, output := range outputs {
        if err := tos.addToBlockOut(ctx, mgr, output,
 ↳ outboundHeight); err != nil {
            return false, err
        }
    }
    return true, nil
}
```

TryAddTxOutItem returns a boolean and an error. When an error occurs, it will always return false as well as an specific kind of error.

x/thorchain/manager_validator_current.go func ragnarokBond

Listing 9: (Line 1)

```
        // refund bond
        txOutItem := TxOutItem{
            Chain:      common.RuneAsset().Chain,
            ToAddress:  na.BondAddress,
            InHash:     common.BlankTxID,
            Coin:       common.NewCoin(common.RuneAsset(), amt),
            Memo:       NewRagnarokMemo(common.BlockHeight(ctx)).
 ↳ String(),
            ModuleName: BondName,
        }
        ok, err := vm.txOutStore.TryAddTxOutItem(ctx, mgr,
 ↳ txOutItem)
        if err != nil {
            if !errors.Is(err, ErrNotEnoughToPayFee) {
                return err
            }
            ok = true
        }
        if !ok {
            continue
        }
        ok, err := vm.txOutStore.TryAddTxOutItem(ctx, mgr,
 ↳ txOutItem)
        if err != nil {
            if !errors.Is(err, ErrNotEnoughToPayFee) {
                return err
            }
            ok = true
        }
        if !ok {
            continue
        }

        // add a pending rangarok transaction
        pending, err := vm.k.GetRagnarokPending(ctx)
        if err != nil {
            return fmt.Errorf("fail to get ragnarok pending: %w", err)
        }
        vm.k.SetRagnarokPending(ctx, pending+1)

        mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)
```

Only one error condition is handled here. This specific error will cause this function to return. Therefore, no refund or slashing will occur.

However, every other error condition is ignored. Although the function will also return a false when other kinds of errors are encountered, this code manually changes the boolean to true.

This makes it so that the ok variable is always true, and therefore the continue instruction will not be executed. The code then continues to a slash event, which removes liquidity units from liquidity providers.

One example scenario is if a user has provided liquidity via a Binance CEX address. This is blacklisted in the TryAddTxOutItem function and causes an error. This error will be ignored by ragnarokBond and a slash will be triggered even though no refund was issued.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Identify all error cases in TryAddTxOutItem and return early with an error message when an error is encountered.

Remediation Plan:

SOLVED: The Maya Protocol team has removed the ragnarokBond feature, so this finding does not apply to recent versions of the codebase.

3.4 (HAL-04) SYNTHETIC ASSETS CAN LOSE COLLATERALIZATION PROTECTION DUE TO AN INTEGER OVERFLOW - LOW

Synthetic assets can be created on Maya as long as there is a sufficient amount of native token backing the synthetic assets. When performing a swap involving a synthetic asset, validation is performed to ensure that the Synthetic asset fits within this boundary.

There is a vulnerability in the calculation of the supply of synthetic assets compared to backing assets. The amount of synthetic assets that exist are compared to a threshold value. If the number of synthetic assets exceeds this value, the swap will not be performed, as this would create a situation where the synthetic assets become undercollateralized. This in turn would allow users to make swaps involving synthetic assets that go outside the safe collateralization range defined by the constants used in Maya chain. This may result in unexpected price changes and liquidity issues for pools containing synthetic assets.

When calculating the number of synthetic assets, the result is converted from an unsigned integer to a signed integer. Therefore, in certain circumstances where the inputs to these variables are very high numbers, the result of the calculation can overflow. If this occurred, the amount of synthetic assets will actually be a very low negative number and so the code will not detect that the synthetic assets have entered into an unsafe range.

The result of this is that the integrity of synthetic assets could be eroded due to a lack of collateralization which may lead to drastic swings in the prices of assets. This could result in loss of funds.

Description:

Code Location:

`x/thorchain/handler_swap.go` validation function Lines 98-114

Listing 10: Calculation of collateralization for synthetic assets

```
98          // fail validation if synth supply is already too high, relative to pool depth
99          maxSynths, err := h.mgr.Keeper().GetMimir(ctx, constants.MaxSynthPerAssetDepth.String())
100         if maxSynths < 0 || err != nil {
101             maxSynths = h.mgr.GetConstants().GetInt64Value(constants.MaxSynthPerAssetDepth)
102         }
103         synthSupply := h.mgr.Keeper().GetTotalSupply(ctx, target.GetSyntheticAsset())
104         pool, err := h.mgr.Keeper().GetPool(ctx, target)
105         if err != nil {
106             return ErrInternal(err, "fail to get pool")
107         }
108         if pool.BalanceAsset.IsZero() {
109             return fmt.Errorf("pool(%s) has zero asset balance", pool.Asset.String())
110         }
111         coverage := int64(synthSupply.MulUint64(MaxWithdrawBasisPoints).Quo(pool.BalanceAsset).Uint64())
112         if coverage > maxSynths {
113             return fmt.Errorf("synth quantity is too high relative to asset depth of related pool (%d/%d)", coverage, maxSynths)
114         }
```

The values MaxSynthPerAssetDepth/maxSynth and MaxWithdrawBasisPoints are used to bound the supply of synthetic assets so that they have a stable backing. When the supply of synthetic assets is found to be outside that range, the validation fails and the swap should not go through (Line 113).

These variables are constants defined in other locations:
MaxSynthPerAssetDepth == 3300 (constants/constants_v1.go)
MaxWithdrawBasisPoints == 10_000 (x/thorchain/types/msg_withdraw_liquidity.go)

However, this validation can fail due to an integer overflow on line 111. If the expression within the int64() call on the right-hand side of Line 111 evaluates to a very high number, the result will overflow when casted to the signed integer type. This could occur for very large values of synthSupply. coverage will evaluate to a negative number, which will be less than maxSynths. Therefore, the expression on Line 112 will be false and the error on line 113 will not be triggered.

This could occur for very large values of synthSupply.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Short-term, investigate whether it is possible to remove the int64 call from this code. Add unit tests that include very large values of synthSupply and ensure that they don't cause unexpected behavior.

Remediation Plan:

SOLVED: The Maya Protocol team addressed this issue in a merge request which can be viewed at this link. The int64() call has been removed.

3.5 (HAL-05) PER-BLOCK REFUNDS ARE LARGER THAN INTENDED DURING RAGNAROK - LOW

Description:

A Ragnarok occurs when nodes vote to remove support for a chain. In the process, all liquidity providers have their liquidity refunded to them.

Refunds are calculated by taking a validator’s bond amount naBond, multiplying it by a parameter nth which represents many blocks, then finally multiplying it by 10.

There is an additional check to ensure nth does not exceed 10. However, this check also has a side effect of setting nth to 10 when its previous value was 9, likely due to an off-by-one error in the logic.

As nth is used to calculate the refund amount, this will result in a higher reward paid to validators.

Code Location:

Listing 11: Calculation of refunds during Ragnarok (Lines 703,721,722,723,725,732)

```go
691 func (vm *validatorMgrV87) ragnarokBond(ctx cosmos.Context, nth
    ↳ int64, mgr Manager) error {
692     // bond should be returned on the back 10, not the first 10
693     nth -= 10
694     if nth < 1 {
695         return nil
696     }
697
698     nas, err := vm.k.ListValidatorsWithBond(ctx)
699     if err != nil {
700         ctx.Logger().Error("can't get nodes", "error", err)
701         return err
702     }
```

```go
    // nth * 10 == the amount of the bond we want to send
    for _, na := range nas {
        naBond, err := mgr.Keeper().CalcNodeLiquidityBond(ctx, na)
        if naBond.IsZero() {
            continue
        }
        if vm.k.VaultExists(ctx, na.PubKeySet.Secp256k1) {
            ygg, err := vm.k.GetVault(ctx, na.PubKeySet.Secp256k1)
            if err != nil {
                return err
            }
            if ygg.HasFunds() {
                ctx.Logger().Info("skip bond refund due to remaining funds", "node address", na.NodeAddress)
                continue
            }
        }

        if nth >= 9 { // cap at 10
            nth = 10
        }
        amt := naBond.MulUint64(uint64(nth)).QuoUint64(10)

        // refund bond
        txOutItem := TxOutItem{
            Chain:      common.RuneAsset().Chain,
            ToAddress:  na.BondAddress,
            InHash:     common.BlankTxID,
            Coin:       common.NewCoin(common.RuneAsset(), amt),
            Memo:       NewRagnarokMemo(common.BlockHeight(ctx)).String(),
            ModuleName: BondName,
        }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Investigate whether this behavior is intentional. If so, the comment should be corrected. As the Ragnarok scenario is intended to refund all liquidity, it is possible that this error does not have an impact if all funds are intended to be returned. Further investigation is likely required in order to determine whether the logic is functioning as intended.

Remediation Plan:

SOLVED: The Maya Protocol team has removed the ragnarokBond feature, so this finding is not applicable to recent versions of the codebase.

3.6 (HAL-06) OVERLY PERMISSIVE CORS SETTINGS - LOW

Description:

The REST client is using an overly permissive Cross-Origin Resource Sharing policy via the Access-Control-Allow-Origin (ACAO) header. This configuration allows any Origin to issue requests to the node and read the response.

Code Location:

x/thorchain/client/rest/rest.go Lines 91-98

Listing 12

```go
func customCORSHeader() mux.MiddlewareFunc {
    return func(next http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, req *
↳ http.Request) {
            w.Header().Set("Access-Control-Allow-Origin", "*")
            next.ServeHTTP(w, req)
        })
    }
```

Note the ACAO header for the Luum project that is part of the Maya infrastructure. (This is a fork of the Midgard API)
internal/api/api.go Lines 148-154

Listing 13

```go
func corsHandler(h http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r *
↳ http.Request) {
                if !strings.HasPrefix(r.URL.Path, proxiedPrefix) {
                        w.Header().Set("Access-Control-Allow-
↳ Origin", "*")
                }
                h.ServeHTTP(w, r)
        })
```

Taken together, a node operator running these services will allow requests from any Origin.

For example, if a node operator visits a website that the attacker controls, that website can issue requests to the node due to the permissive Access-Control-Allow-Origin policy. This can tie up system resources or expose protected endpoints.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Create an allow-list of approved IP addresses that should be able to interact with the node and the Luum API. Normally, this will be one or more client applications that are presented to end-users.

Ensure that the node uses the best practices outlined in the documentation for the Cosmos SDK. By default, the CORS policies are disabled. The nodes should also be set up using the sentry architecture for distributed denial-of-service (DDoS) protection. Using the sentry architecture, there is no need for a permissive ACAO policy, as only the sentry should communicate with the node.

Remediation Plan:

RISK ACCEPTED: The Maya Protocol team accepted the risk of this finding.

3.7 (HAL-07) FUNCTION SETMIMIR FAILS SILENTLY WHEN DISABLED - LOW

Description:

The Mimir module is responsible for setting key values in the Maya protocol. It is overridden when the ReleaseTheKraken setting is enabled. However, there is no error message informing an admin that the SetMimir command has failed.

If one admin has enabled ReleaseTheKraken, a second admin who tries to change a setting via SetMimir will be unable to do so, but they will not be informed. For example, if this second admin wishes to halt the chain by enabling the NodePauseChainGlobal setting, this action will fail, and they will not know. Trades will continue even though this admin wished to disable them.

Code Location:

x/thorchain/keeper/v1/keeper_mimir.go Lines 46-52

Listing 14

```
46 // SetMimir save a mimir value to key value store
47 func (k KVStore) SetMimir(ctx cosmos.Context, key string, value
   ↳ int64) {
48     // if we have the kraken, mimir is no more, ignore him
49     if k.haveKraken(ctx) {
50         return
51     }
52     k.setInt64(ctx, k.GetKey(ctx, prefixMimir, key), value)
53 }
```

This code is called by the function handleV87 in x/thorchain/handler_node_pause_chain.go Lines 123-130

Listing 15

```
    key := "NodePauseChainGlobal"
    h.mgr.Keeper().SetMimir(ctx, key, pauseHeight)
    mimirEvent := NewEventSetMimir(strings.ToUpper(key), strconv.
 ↳ FormatInt(pauseHeight, 10))
    if err := h.mgr.EventMgr().EmitEvent(ctx, mimirEvent); err !=
 ↳ nil {
        ctx.Logger().Error("fail to emit set_mimir event", "error"
 ↳ , err)
    }

    return nil
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Add an error message when SetMimir is called and ReleaseTheKraken is active. This will prevent confusion for node administrators.

Note that there are other examples in the codebase where SetMimir is called. The example of pausing the chain is just one example.

Remediation Plan:

SOLVED: The Maya Protocol team has added an error message when “ReleaseTheKraken” is active.

3.8 (HAL-08) NO COMPLEXITY REQUIREMENTS FOR BIFROST PASSPHRASE - LOW

Description:

The Bifrost functionality in Maya uses a LevelDB instance in order to store data. When creating a new database instance, there is no enforcement of passphrase strength. As a result, an administrator can use a weak password.

Code Location:

Listing 16: A new LevelDB instance can be created with a weak password

```go
// NewSignerStore create a new instance of SignerStore. If no
↳ folder is given,
// an in memory implementation is used.
func NewSignerStore(levelDbFolder, passphrase string) (*
↳ SignerStore, error) {
    var db *leveldb.DB
    var err error
    if len(levelDbFolder) == 0 {
        log.Warn().Msg("level db folder is empty, create in memory
↳ storage")
        // no directory given, use in memory store
        storage := storage.NewMemStorage()
        db, err = leveldb.Open(storage, nil)
        if err != nil {
            return nil, fmt.Errorf("fail to in memory open level
↳ db: %w", err)
        }
    } else {
        db, err = leveldb.OpenFile(levelDbFolder, nil)
        if err != nil {
            return nil, fmt.Errorf("fail to open level db %s: %w",
↳ levelDbFolder, err)
        }
    }
    levelDbStorage, err := blockscanner.NewLevelDBScannerStorage(
↳ db)
    if err != nil {
        return nil, errors.New("fail to create level db")
    }
    return &SignerStore{
        LevelDBScannerStorage: levelDbStorage,
        logger:                log.With().Str("module", "signer-
↳ storage").Logger(),
        db:                    db,
        passphrase:            passphrase,
    }, nil
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Ensure that passphrases contain a high degree of entropy. It is recommended to use long, unique passwords in order to prevent an attacker from guessing the password.

Remediation Plan:

SOLVED: The Maya Protocol team has added password complexity rules in a recent merge request.

3.9 (HAL-09) USE OF INSECURE HASH FUNCTION MD5 - LOW

Description:

The MD5 hash function is considered unsuitable for security purposes due to multiple issues in the algorithm.

Code Location:

common/encryption.go

Listing 17: (Lines 10,14)

```
 3 import (
 4     "crypto/aes"
 5     "crypto/cipher"
 6     "crypto/rand"
 7     "encoding/hex"
 8     "io"
 9
10     "crypto/md5" // nolint
11 )
12
13 func createHash(key string) (string, error) {
14     hasher := md5.New() // nolint
15     _, err := hasher.Write([]byte(key))
16     return hex.EncodeToString(hasher.Sum(nil)), err
17 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Replace MD5 with a modern, secure equivalent such as SHA256.

Remediation Plan:

RISK ACCEPTED: The Maya Protocol team accepted the risk of this finding.

3.10 (HAL-10) LACK OF KEY-DERIVATION FUNCTION USED FOR AES ENCRYPTION KEY - LOW

Description:

A key derivation function (KDF) can be used to take a value with low entropy and convert it to a value with high-entropy. Commonly, it is used to take an input like a password and generate an encryption degree from it. This key can then be used as the input to an encryption algorithm like AES.

Code Location:

common/encryption.go

Listing 18: (Line 1)

```go
// Encrypt the input data with passphrase
func Encrypt(data []byte, passphrase string) ([]byte, error) {
    hash, err := createHash(passphrase)
    if err != nil {
        return nil, err
    }

    block, _ := aes.NewCipher([]byte(hash))
    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return nil, err
    }

    nonce := make([]byte, gcm.NonceSize())
    if _, err = io.ReadFull(rand.Reader, nonce); err != nil {
        return nil, err
    }

    ciphertext := gcm.Seal(nonce, nonce, data, nil)
    return ciphertext, nil
}
```

```go

// Decrypt the input data with passphrase
func Decrypt(data []byte, passphrase string) ([]byte, error) {
    hash, err := createHash(passphrase)
    if err != nil {
        return nil, err
    }

    key := []byte(hash)
    block, err := aes.NewCipher(key)
    if err != nil {
        return nil, err
    }

    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return nil, err
    }
    nonceSize := gcm.NonceSize()
    nonce, ciphertext := data[:nonceSize], data[nonceSize:]
    plaintext, err := gcm.Open(nil, nonce, ciphertext, nil)
    if err != nil {
        return nil, err
    }
    return plaintext, nil
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Incorporate the use of a KDF for the AES key rather than use a passphrase.

Remediation Plan:

RISK ACCEPTED: The Maya Protocol team accepted the risk of finding.

3.11 (HAL-11) LOGIC PROBLEMS CAN OCCUR WHEN CONVERTING MIMIR VALUES BETWEEN SIGNED AND UNSIGNED INTEGERS - LOW

Description:

The Mimir module controls the configuration of constants used within the protocol. In numerous locations in the codebase, these values are retrieved as uint64 types and then casted to int64 values.
If an attacker is able to control these parameters, or if the parameters themselves are set to very high values during normal operation, an integer overflow will occur. In this case, very high numbers will become negative numbers. On the other hand, negative Mimir values when casted to unsigned integers will become very large positive numbers.

When these values are converted to int from uint or vice-versa and then are evaluated in conditional statements, unexpected and dangerous behavior can occur.

The following example shows one location where this could be problematic.

Code Location:

Listing 19: (Lines 91,94,95,100)

```
89 nativeTxFee, err := h.mgr.Keeper().GetMimir(ctx, constants.
 ↳ NativeTransactionFee.String())
90     if err != nil || nativeTxFee < 0 {
91         nativeTxFee = h.mgr.GetConstants().GetInt64Value(constants
 ↳ .NativeTransactionFee)
92     }
93
94     gas := common.NewCoin(common.RuneNative, cosmos.NewUint(uint64
 ↳ (nativeTxFee)))
95     gasFee, err := gas.Native()
96     if err != nil {
        return nil, ErrInternal(err, "fail to get gas fee")
    }

    totalCoins := cosmos.NewCoins(gasFee).Add(msg.Amount...)

    if !h.mgr.Keeper().HasCoins(ctx, msg.FromAddress, totalCoins)
↳ {
        return nil, cosmos.ErrInsufficientCoins(err, "insufficient
↳  funds")
    }
```

In the above example, if the NativeTransactionFee is a negative number, it will be cast to a large positive number in this code. This will cause gas costs to become extremely high. In this situation, users will not be able to perform transactions because the price of gas will be too high.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

We recommend that the appropriate verification is done before the use of the Mimir parameters. When using signed integers, there should be error-handling in the case that they are negative.

Long-term, consider whether it is possible to use unsigned integers instead of signed integers in order to eliminate the risk of overflows during casting operations.

Remediation Plan:

RISK ACCEPTED: The Maya Protocol team accepted the risk of finding.

3.12 (HAL-12) UNHANDLED ERRORS - LOW

Description:

In addition to the detailed issues reported above, there are multiple additional locations in the codebase where errors are not being handled.

For example, there is no error checking being done on the functions SlashNodeAccountLP and removeBondAddress that are being called from handler_ban.go, handler_unbond.go, helpers.go, manager_validator_current.go, manager_validator_v80.go and manager_validator_84.go.

Code Location:

Listing 20

```
[/manager_validator_v84.go]
 759: mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)

[/manager_validator_v80.go]
 761: mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)

[/manager_validator_current.go]
 752: mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)

[/helpers.go]
 325: mgr.Slasher().SlashNodeAccountLP(ctx, *nodeAcc, slashRune)

[/handler_unbond.go]
 235: removeBondAddress(ctx, h.mgr, common.Address(provider.
↳ BondAddress.String()))
 243: removeBondAddress(ctx, h.mgr, common.Address(msg.
↳ BondProviderAddress.String()))
 267: removeBondAddress(ctx, h.mgr, common.Address(from.String()))

[/handler_ban.go]
 125: h.mgr.Slasher().SlashNodeAccountLP(ctx, banner, slashAmount)

```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

We recommend that the appropriate error checking be implemented to avoid unexpected behavior or crashes. The use of an automated tool like errcheck can help identify instances where errors are not handled.

Remediation Plan:

SOLVED: The Maya Protocol team has added additional error handling in a recent merge request.

3.13 (HAL-13) USE OF OUTDATED COSMOS SDK VERSION - LOW

Description:

The CosmosSDK version used by the project is 0.45.2 which is out-of-date. Newer versions of CosmosSDK contain important performance increases and bug fixes.

Below is a non-exhaustive list of bugs that have been fixed in recent releases:

Version 0.46.2
- (grpc) #13417 fix grpc query panic that could crash the node (backport #13352).

Version 0.46.0
- #11969 Fix the panic error in x/upgrade when AppVersion is not set.
- (types) #9627 Fix nil pointer panic on NewBigIntFromInt
- (x/upgrade) #10189 Removed potential sources of non-determinism in upgrades

Version 0.45.7
- (simapp) #12437 fix the non-determinstic behavior in simulations caused by GenTx and check empty coins slice before it is used to create banktype.MsgSend.

Version 0.45.5
- #11772 Limit types.Dec length to avoid overflow.

Version 0.45.4
- #11724 Fix data race issues with api.Server.

Version 0.45.2
- (store) #11117 Fix data race in store trace component

Code Location:

go.mod

Listing 21: CosmosSDK version in use based on go.mod

```
14:     github.com/cosmos/cosmos-sdk v0.45.1
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider upgrading the CosmosSDK version used by the project. Note that upgrades may introduce breaking changes.
Further details can be found in the CosmosSDK change log.

Remediation Plan:

SOLVED: The Maya Protocol team has updated the CosmosSDK version to version 0.45.9 in a recent merge request.

3.14 (HAL-14) VULNERABLE THIRD PARTY PACKAGES - LOW

Description:

During the audit, Halborn identified installed 3rd party packages that contain known security vulnerabilities.

Packages:

ID	Package	Rating	Description
CVE-2022-21698	client_golang	HIGH	Denial of Service attack
sonatype-2022-3945	go-buffer-pool	MEDIUM	Integer Overflow or Wraparound
CVE-2021-0076	go-ethereum	HIGH	Uncontrolled Resource Consumption
CVE-2022-29177	go-ethereum	MEDIUM	Uncontrolled Resource Consumption
CVE-2022-37450	go-ethereum	MEDIUM	Improper Input Validation

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to keep all installed third-party packages up to date and apply all security fixes applied.

Remediation Plan:

SOLVED: The Maya Protocol team updated vulnerable packages in a recent merge request.

3.15 (HAL-15) USE OF DEPRECATED FUNCTIONS - INFORMATIONAL

Description:

Deprecated Go functions were identified in the codebase. The function(s) are:

- **ioutil** package
- **strings.Title**

Code Location:

Listing 22: Uses of ioutil

```
x/thorchain/client/rest/broadcast.go
5:       "io/ioutil"
37:              body, err := ioutil.ReadAll(r.Body)

bifrost/pkg/chainclients/binance/binance_block_scanner.go
10:      "io/ioutil"
116:     data, err := ioutil.ReadAll(resp.Body)
151:     bz, err := ioutil.ReadAll(resp.Body)
275:     buf, err := ioutil.ReadAll(resp.Body)

bifrost/pkg/chainclients/binance/binance.go
9:       "io/ioutil"
191:     data, err := ioutil.ReadAll(resp.Body)
475:     body, err := ioutil.ReadAll(resp.Body)
525:     body, err := ioutil.ReadAll(resp.Body)

x/thorchain/types/test_common.go
5:       "io/ioutil"
240:     dat, err := ioutil.ReadFile(path.Join(dir, "version"))

common/node-relay.go
10:      "io/ioutil"
47:      body, err := ioutil.ReadAll(resp.Body)
107:     body, err := ioutil.ReadAll(resp.Body)
```

```
bifrost/thorclient/thorchain.go
7:       "io/ioutil"
166:     buf, err := ioutil.ReadAll(resp.Body)

cmd/bifrost/main.go
9:       "io/ioutil"
264:     buf, err := ioutil.ReadFile(file)
```

Listing 23: Uses of strings.Title

```
x/thorchain/types/type_node_account.go
19:      if _, ok := NodeStatus_value[strings.Title(x.String())]; !
↳ ok {
42:      if val, ok := NodeStatus_value[strings.Title(ps)]; ok {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Replace deprecated functions with up-to-date functions:

- ioutil can be replaced by package io or os.
- strings.Title can be replaced with the function Title in the cases package.

Remediation Plan:

SOLVED: The Maya Protocol team has replaced the deprecated functions in a recent merge request.

3.16 (HAL-16) PANIC IS USED FOR ERROR HANDLING - INFORMATIONAL

Description:

Several instances of the panic function were identified in the codebase. They appear to be used to handle errors. This can cause potential issues, as invoking a panic can cause the program to halt execution and crash in some cases. This in turn can negatively impact the availability of the software for users.

Code Location:

Listing 24: Instances of panic identified in the codebase

```
x/thorchain/memo/memo.go:193:            err = fmt.Errorf("panicked
↳  parsing memo(%s), err: %s", memo, r)
x/thorchain/memo/memo.go:263:            err = fmt.Errorf("panicked
↳  parsing memo(%s), err: %s", memo, r)
x/thorchain/module.go:105:       panic(err)
x/thorchain/client/rest/stdtx.go:57:         panic(err)
x/thorchain/client/rest/rest.go:84:      panic(err)
x/thorchain/helpers.go:164://    // Should never happen, but this
↳ prevents a divide-by-zero panic in case it does
x/thorchain/helpers.go:946: // capture panics
x/thorchain/helpers.go:949:         ctx.Logger().Error("panic
↳ while emitting end block telemetry", "error", err)
x/thorchain/genesis.go:135:         panic(err)
x/thorchain/genesis.go:150:             panic(err)
x/thorchain/genesis.go:156:         // we should panic
x/thorchain/genesis.go:157:         panic(err)
x/thorchain/genesis.go:163:         panic(err)
x/thorchain/genesis.go:169:         panic(err)
x/thorchain/genesis.go:184:         panic(err)
x/thorchain/genesis.go:190:         panic(err)
x/thorchain/genesis.go:197:         panic(err)
x/thorchain/genesis.go:200:         panic(err)
x/thorchain/genesis.go:204:     panic(err)
x/thorchain/genesis.go:209:         panic(err)
x/thorchain/genesis.go:214:         panic(err)
```

```
x/thorchain/genesis.go:230:         panic(err)
x/thorchain/genesis.go:233:         panic(err)
x/thorchain/genesis.go:241:         panic(err)
x/thorchain/genesis.go:244:         panic(err)
x/thorchain/genesis.go:251:         panic(err)
x/thorchain/genesis.go:258:         panic(err)
x/thorchain/genesis.go:261:         panic(err)
x/thorchain/genesis.go:350:         panic(err)
x/thorchain/genesis.go:391:     panic(err)
x/thorchain/genesis.go:396:     panic(err)
x/thorchain/genesis.go:412:     panic(err)
bifrost/tss/keygen.go:100:      panic("tss keygen timeout")
bifrost/pkg/chainclients/terra/wasm/msgs.go:34:     panic(err)
x/thorchain/types/test_common.go:128:       panic(err)
x/thorchain/types/test_common.go:242:       panic(err)
x/thorchain/types/test_common.go:246:       panic(err)
bifrost/thorclient/keys.go:86:      panic(err)
bifrost/thorclient/thorchain.go:143:        panic(err)
bifrost/signer/sign.go:217:                         panic(fmt.Errorf("tx out item: %+v , keysign timeout : %w", item.TxOutItem, err))
common/type_convert.go:25:// GetShare this method will panic if any of the input parameter can't be convert to cosmos.Dec
common/type_convert.go:40:      panic(fmt.Errorf("fail to convert %s to cosmos.Dec: %w", allocation.String(), err))
common/type_convert.go:45:      panic(fmt.Errorf("fatil to convert %s to cosmos.Dec: %w", part.String(), err))
common/type_convert.go:49:      panic(fmt.Errorf("fail to convert%s to cosmos.Dec: %w", total.String(), err))
common/type_convert.go:57:// SafeSub subtract input2 from input1, given cosmos.Uint can't be negative , otherwise it will panic
app/export.go:81:           panic("expected validator, not found")
cmd/thornode/cmd/ed25519_keys.go:158:           panic("it should not fail")
tools/analyze/main.go:35:           panic("node was not *ast.File")
tools/analyze/main.go:48:           panic("node was not *ast.RangeStmt")
cmd/thornode/cmd/root.go:183:       panic(err)
cmd/thornode/cmd/root.go:187:       panic(err)
tools/pubkey2address/pubkey2address.go:16:      panic("no pubkey provided")
tools/pubkey2address/pubkey2address.go:41:      panic(err)
tools/pubkey2address/pubkey2address.go:54:          panic(err)
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Instead of using panics, custom errors should be defined and handled according to the Cosmos best practices.

Remediation Plan:

ACKNOWLEDGED: The Maya Protocol team acknowledged this finding.

3.17 (HAL-17) OPEN TODOS IN CODEBASE - INFORMATIONAL

Description:

Open To-dos can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

Listing 25: Open Todos

```
manager_yggdrasil_current.go:117:    // TODO: We are assuming here
↳ that the pub key is Secp256K1
manager_network_v76.go:474:          // TODO: need to leave this
↳ here so we don't cause a consensus failure.
manager_slasher_current.go:289:      // TODO add split logic to
↳ send it out from multiple asgards in
manager_txout_v83.go:396:            // TODO(leifthelucky): "
↳ runeFee" is ultimately deducted from the pool
withdraw_current.go:125:             // TODO: chain specific logic
↳ should be in a single location
handler_tss.go:294:                  // TODO(@cizin): Subtract bond
↳  from lp assets also.
swap_current_test.go:527:            // TODO(leifthelucky): The
↳ total synth supply doesn't actually change. This is very puzzling.
withdraw_v84.go:157:                 // TODO: chain specific logic
↳ should be in a single location
managers_dummy_test.go:35:           // TODO add dummy swap queue
managers_dummy_test.go:51:           // TODO add dummy swap queue
withdraw_v76.go:132:                 // TODO: chain specific logic
↳ should be in a single location
handler_withdraw.go:149:             // TODO: chain specific logic
↳ should be in a single location
handler_ban.go:116:                  // TODO(@cizin): Instead of
↳ moving it to reserve, slash anti-lps.
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving the To-dos before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

ACKNOWLEDGED: The Maya Protocol team acknowledged this finding.

3.18 (HAL-18) SPELLING MISTAKES IN THE CODEBASE - INFORMATIONAL

Description:

Spelling mistakes were identified within the codebase.

Code Location:

unmashsl

Listing 26

```
x/thorchain/manager_network_v87.go
735:                    return nil, cosmos.ZeroUint(), fmt.Errorf
↳ ("fail to unmarhsl pool: %w", err)

x/thorchain/manager_network_current.go
720:                    return nil, cosmos.ZeroUint(), fmt.Errorf
↳ ("fail to unmarhsl pool: %w", err)

x/thorchain/manager_network_v76.go
763:                    return nil, cosmos.ZeroUint(), fmt.Errorf
↳ ("fail to unmarhsl pool: %w", err)

`x/thorchain/manager_network_current.go`
720             return nil, cosmos.ZeroUint(), fmt.Errorf("fail to
↳  unmarhsl pool: %w", err)
```

blockPerDar -- x/thorchain/manager_gas_current.go

Listing 27

```
204:    blockPerDar := gm.constantsAccessor.GetInt64Value(
↳ constants.BlocksPerDay)
205:    if IsPeriodLastBlock(ctx, uint64(blockPerDar)) {
```

minimun, need it to -- constants/constants_v1.go

Listing 28

```
70                MinRuneForMayaFundDist:          100000000,
            // The minimun amount of tokens need it to distribute on
MayaFund
```

Listing 29

```
x/thorchain/handler_add_liquidity.go:181:24: `substract` is a
misspelling of `subtract` (misspell)
  38         // Avoid overflow and substract the liquidity that is
being used for bond to get Node Exclusive Liquidity (NEL)
```

impermament --> impermanent -- x/thorchain/withdraw_current.go

Listing 30

```
71      // calculate any impermament loss protection or not
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that all filenames and usage of words within the code are spelled correctly, as this will avoid confusion during development.

Proper spelling can also help convey a sense of professionalism to various project stakeholders.

Remediation Plan:

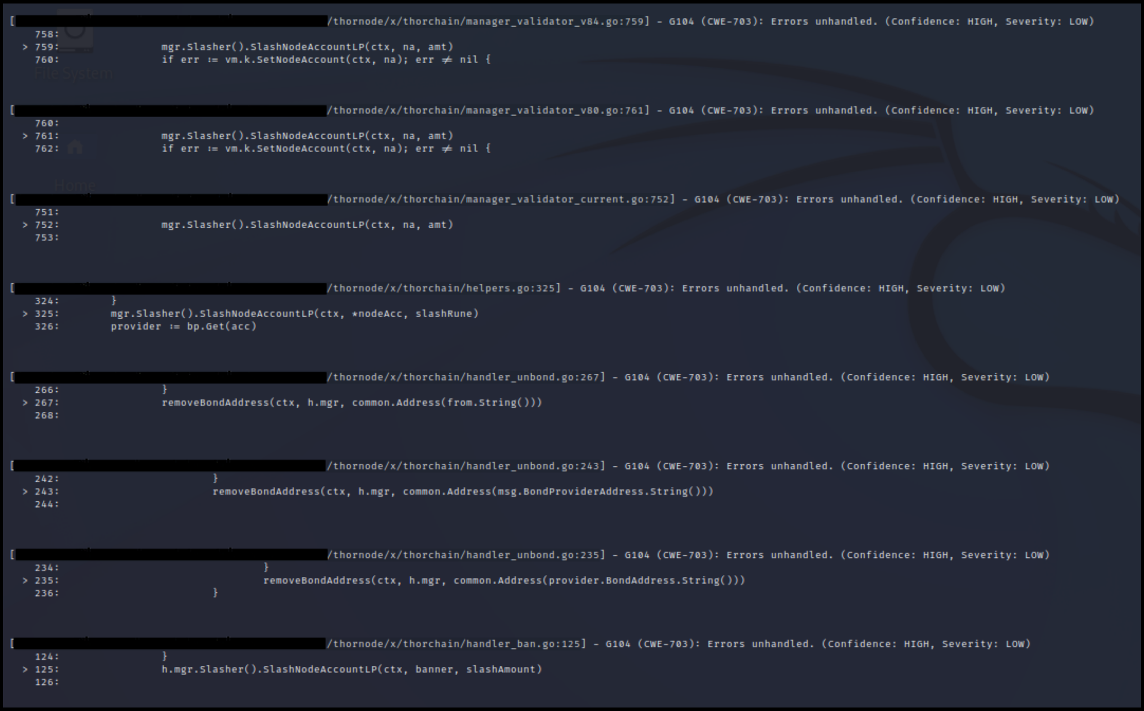
SOLVED: The Maya Protocol team has resolved these spelling mistakes in a recent merge request.

AUTOMATED TESTING

Description:

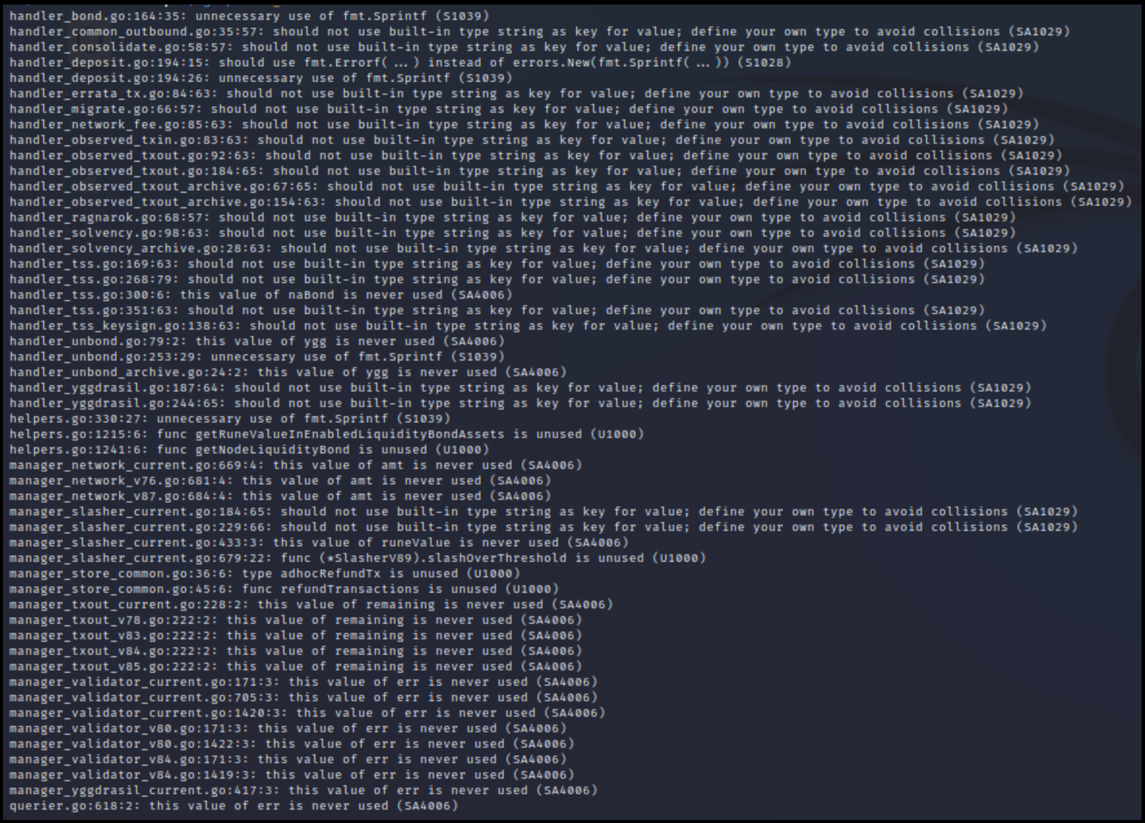
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped component. Among the tools used were staticcheck, gosec, unconvert and Nancy. After Halborn verified all the contracts and scoped structures in the repository and was able to compile them correctly, these tools were leveraged on scoped structures. With these tools, Halborn can statically verify security related issues across the entire codebase.

Gosec - Security Analysis Output Sample:



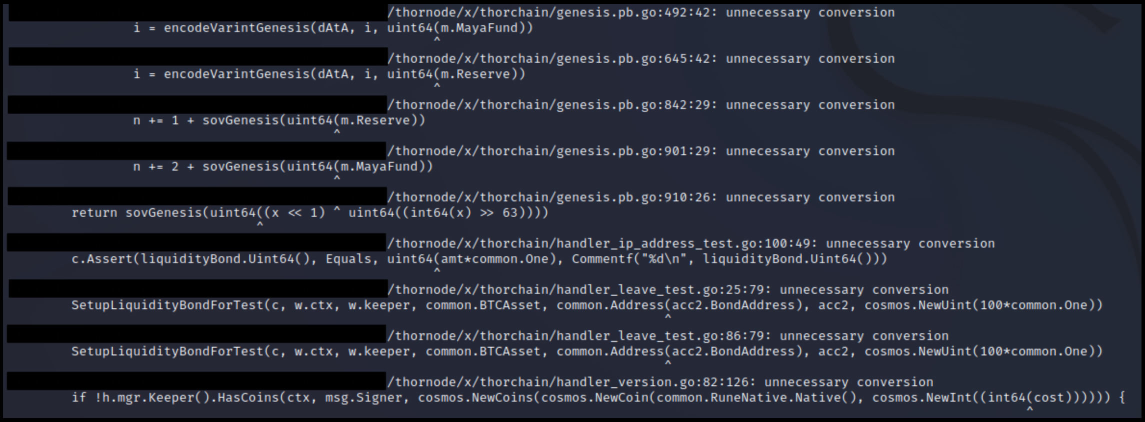
```
[[redacted]/thornode/x/thorchain/manager_validator_v84.go:759] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   758:
 > 759:            mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)
   760:            if err := vm.k.SetNodeAccount(ctx, na); err ≠ nil {


[[redacted]/thornode/x/thorchain/manager_validator_v80.go:761] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   760:
 > 761:            mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)
   762:            if err := vm.k.SetNodeAccount(ctx, na); err ≠ nil {


[[redacted]/thornode/x/thorchain/manager_validator_current.go:752] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   751:
 > 752:            mgr.Slasher().SlashNodeAccountLP(ctx, na, amt)
   753:


[[redacted]/thornode/x/thorchain/helpers.go:325] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   324:        }
 > 325:        mgr.Slasher().SlashNodeAccountLP(ctx, *nodeAcc, slashRune)
   326:        provider := bp.Get(acc)


[[redacted]/thornode/x/thorchain/handler_unbond.go:267] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   266:            }
 > 267:            removeBondAddress(ctx, h.mgr, common.Address(from.String()))
   268:


[[redacted]/thornode/x/thorchain/handler_unbond.go:243] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   242:                    }
 > 243:                    removeBondAddress(ctx, h.mgr, common.Address(msg.BondProviderAddress.String()))
   244:


[[redacted]/thornode/x/thorchain/handler_unbond.go:235] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   234:                            }
 > 235:                            removeBondAddress(ctx, h.mgr, common.Address(provider.BondAddress.String()))
   236:                    }


[[redacted]/thornode/x/thorchain/handler_ban.go:125] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
   124:            }
 > 125:            h.mgr.Slasher().SlashNodeAccountLP(ctx, banner, slashAmount)
   126:
```

Staticcheck - Security Analysis Output Sample:



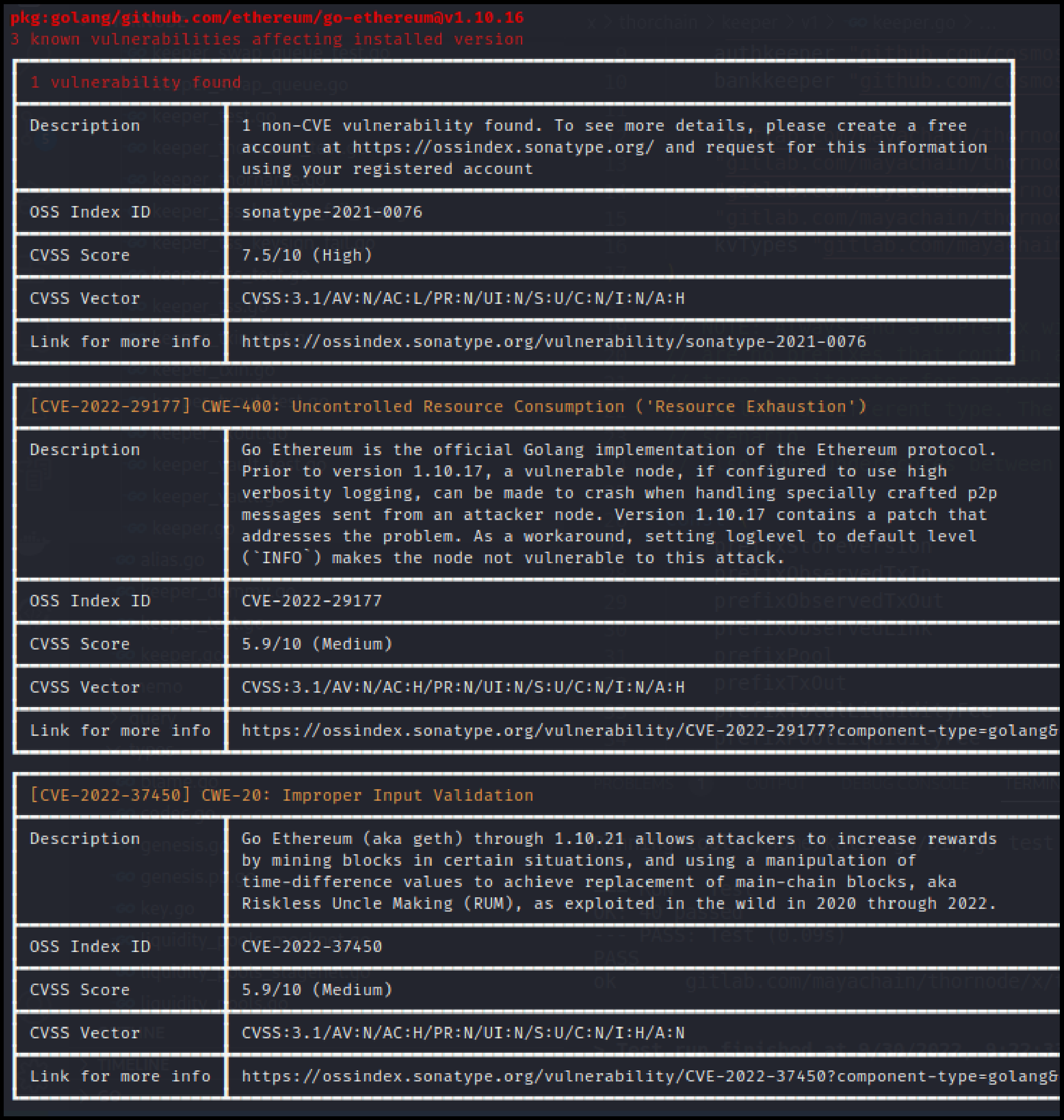
```
handler_bond.go:164:35: unnecessary use of fmt.Sprintf (S1039)
handler_common_outbound.go:35:57: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_consolidate.go:58:57: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_deposit.go:194:15: should use fmt.Errorf( … ) instead of errors.New(fmt.Sprintf( … )) (S1028)
handler_deposit.go:194:26: unnecessary use of fmt.Sprintf (S1039)
handler_errata_tx.go:84:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_migrate.go:66:57: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_network_fee.go:85:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_observed_txin.go:83:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_observed_txout.go:92:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_observed_txout.go:184:65: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_observed_txout_archive.go:67:65: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_observed_txout_archive.go:154:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_ragnarok.go:68:57: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_solvency.go:98:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_solvency_archive.go:28:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_tss.go:169:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_tss.go:268:79: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_tss.go:300:6: this value of naBond is never used (SA4006)
handler_tss.go:351:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_tss_keysign.go:138:63: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_unbond.go:79:2: this value of ygg is never used (SA4006)
handler_unbond.go:253:29: unnecessary use of fmt.Sprintf (S1039)
handler_unbond_archive.go:24:2: this value of ygg is never used (SA4006)
handler_yggdrasil.go:187:64: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
handler_yggdrasil.go:244:65: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
helpers.go:330:27: unnecessary use of fmt.Sprintf (S1039)
helpers.go:1215:6: func getRuneValueInEnabledLiquidityBondAssets is unused (U1000)
helpers.go:1241:6: func getNodeLiquidityBond is unused (U1000)
manager_network_current.go:669:4: this value of amt is never used (SA4006)
manager_network_v76.go:681:4: this value of amt is never used (SA4006)
manager_network_v87.go:684:4: this value of amt is never used (SA4006)
manager_slasher_current.go:184:65: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
manager_slasher_current.go:229:66: should not use built-in type string as key for value; define your own type to avoid collisions (SA1029)
manager_slasher_current.go:433:3: this value of runeValue is never used (SA4006)
manager_slasher_current.go:679:22: func (*SlasherV89).slashOverThreshold is unused (U1000)
manager_store_common.go:36:6: type adhocRefundTx is unused (U1000)
manager_store_common.go:45:6: func refundTransactions is unused (U1000)
manager_txout_current.go:228:2: this value of remaining is never used (SA4006)
manager_txout_v78.go:222:2: this value of remaining is never used (SA4006)
manager_txout_v83.go:222:2: this value of remaining is never used (SA4006)
manager_txout_v84.go:222:2: this value of remaining is never used (SA4006)
manager_txout_v85.go:222:2: this value of remaining is never used (SA4006)
manager_validator_current.go:171:3: this value of err is never used (SA4006)
manager_validator_current.go:705:3: this value of err is never used (SA4006)
manager_validator_current.go:1420:3: this value of err is never used (SA4006)
manager_validator_v80.go:171:3: this value of err is never used (SA4006)
manager_validator_v80.go:1422:3: this value of err is never used (SA4006)
manager_validator_v84.go:171:3: this value of err is never used (SA4006)
manager_validator_v84.go:1419:3: this value of err is never used (SA4006)
manager_yggdrasil_current.go:417:3: this value of err is never used (SA4006)
querier.go:618:2: this value of err is never used (SA4006)
```

Unconvert - Security Analysis Output Sample:

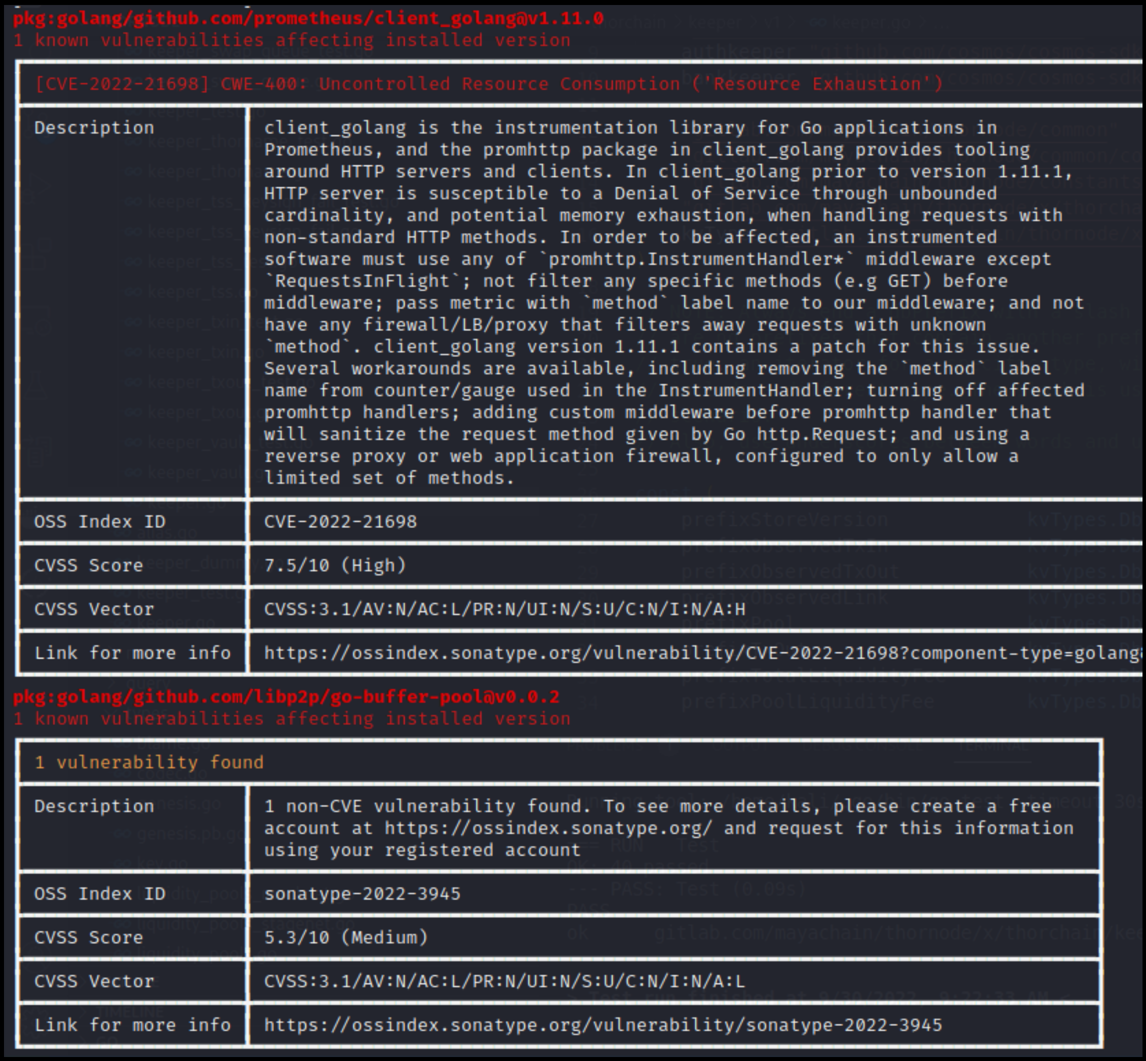


```
[redacted]/thornode/x/thorchain/genesis.pb.go:492:42: unnecessary conversion
                i = encodeVarintGenesis(dAtA, i, uint64(m.MayaFund))
                                                       ^
[redacted]/thornode/x/thorchain/genesis.pb.go:645:42: unnecessary conversion
                i = encodeVarintGenesis(dAtA, i, uint64(m.Reserve))
                                                       ^
[redacted]/thornode/x/thorchain/genesis.pb.go:842:29: unnecessary conversion
                n += 1 + sovGenesis(uint64(m.Reserve))
                                          ^
[redacted]/thornode/x/thorchain/genesis.pb.go:901:29: unnecessary conversion
                n += 2 + sovGenesis(uint64(m.MayaFund))
                                          ^
[redacted]/thornode/x/thorchain/genesis.pb.go:910:26: unnecessary conversion
        return sovGenesis(uint64((x << 1) ^ uint64((int64(x) >> 63))))
                                ^
[redacted]/thornode/x/thorchain/handler_ip_address_test.go:100:49: unnecessary conversion
        c.Assert(liquidityBond.Uint64(), Equals, uint64(amt*common.One), Commentf("%d\n", liquidityBond.Uint64()))
                                                       ^
[redacted]/thornode/x/thorchain/handler_leave_test.go:25:79: unnecessary conversion
        SetupLiquidityBondForTest(c, w.ctx, w.keeper, common.BTCAsset, common.Address(acc2.BondAddress), acc2, cosmos.NewUint(100*common.One))
                                                                                     ^
[redacted]/thornode/x/thorchain/handler_leave_test.go:86:79: unnecessary conversion
        SetupLiquidityBondForTest(c, w.ctx, w.keeper, common.BTCAsset, common.Address(acc2.BondAddress), acc2, cosmos.NewUint(100*common.One))
                                                                                     ^
[redacted]/thornode/x/thorchain/handler_version.go:82:126: unnecessary conversion
        if !h.mgr.Keeper().HasCoins(ctx, msg.Signer, cosmos.NewCoins(cosmos.NewCoin(common.RuneNative.Native(), cosmos.NewInt((int64(cost)))))) {
                                                                                                                            ^
```

Nancy - Security Analysis Output Sample:



```
pkg:golang/github.com/ethereum/go-ethereum@v1.10.16
3 known vulnerabilities affecting installed version
┌────────────────────────────────────────────────────────────────────────────────────────────┐
│ 1 vulnerability found                                                                      │
├────────────────────┬───────────────────────────────────────────────────────────────────────┤
│ Description        │ 1 non-CVE vulnerability found. To see more details, please create a free │
│                    │ account at https://ossindex.sonatype.org/ and request for this information │
│                    │ using your registered account                                         │
├────────────────────┼───────────────────────────────────────────────────────────────────────┤
│ OSS Index ID       │ sonatype-2021-0076                                                    │
├────────────────────┼───────────────────────────────────────────────────────────────────────┤
│ CVSS Score         │ 7.5/10 (High)                                                         │
├────────────────────┼───────────────────────────────────────────────────────────────────────┤
│ CVSS Vector        │ CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H                          │
├────────────────────┼───────────────────────────────────────────────────────────────────────┤
│ Link for more info │ https://ossindex.sonatype.org/vulnerability/sonatype-2021-0076        │
└────────────────────┴───────────────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────────────────────────────────────
│ [CVE-2022-29177] CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')
├────────────────────┬─────────────────────────────────────────────────────────────────────────
│ Description        │ Go Ethereum is the official Golang implementation of the Ethereum protocol.
│                    │ Prior to version 1.10.17, a vulnerable node, if configured to use high
│                    │ verbosity logging, can be made to crash when handling specially crafted p2p
│                    │ messages sent from an attacker node. Version 1.10.17 contains a patch that
│                    │ addresses the problem. As a workaround, setting loglevel to default level
│                    │ (`INFO`) makes the node not vulnerable to this attack.
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ OSS Index ID       │ CVE-2022-29177
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ CVSS Score         │ 5.9/10 (Medium)
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ CVSS Vector        │ CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ Link for more info │ https://ossindex.sonatype.org/vulnerability/CVE-2022-29177?component-type=golang&
└────────────────────┴─────────────────────────────────────────────────────────────────────────

┌──────────────────────────────────────────────────────────────────────────────────────────────
│ [CVE-2022-37450] CWE-20: Improper Input Validation
├────────────────────┬─────────────────────────────────────────────────────────────────────────
│ Description        │ Go Ethereum (aka geth) through 1.10.21 allows attackers to increase rewards
│                    │ by mining blocks in certain situations, and using a manipulation of
│                    │ time-difference values to achieve replacement of main-chain blocks, aka
│                    │ Riskless Uncle Making (RUM), as exploited in the wild in 2020 through 2022.
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ OSS Index ID       │ CVE-2022-37450
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ CVSS Score         │ 5.9/10 (Medium)
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ CVSS Vector        │ CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N
├────────────────────┼─────────────────────────────────────────────────────────────────────────
│ Link for more info │ https://ossindex.sonatype.org/vulnerability/CVE-2022-37450?component-type=golang&
└────────────────────┴─────────────────────────────────────────────────────────────────────────
```



```
pkg:golang/github.com/prometheus/client_golang@v1.11.0
1 known vulnerabilities affecting installed version
┌──────────────────────────────────────────────────────────────────────────────────
│ [CVE-2022-21698] CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')
├──────────────────┬───────────────────────────────────────────────────────────────
│ Description      │ client_golang is the instrumentation library for Go applications in
│                  │ Prometheus, and the promhttp package in client_golang provides tooling
│                  │ around HTTP servers and clients. In client_golang prior to version 1.11.1,
│                  │ HTTP server is susceptible to a Denial of Service through unbounded
│                  │ cardinality, and potential memory exhaustion, when handling requests with
│                  │ non-standard HTTP methods. In order to be affected, an instrumented
│                  │ software must use any of `promhttp.InstrumentHandler*` middleware except
│                  │ `RequestsInFlight`; not filter any specific methods (e.g GET) before
│                  │ middleware; pass metric with `method` label name to our middleware; and not
│                  │ have any firewall/LB/proxy that filters away requests with unknown
│                  │ `method`. client_golang version 1.11.1 contains a patch for this issue.
│                  │ Several workarounds are available, including removing the `method` label
│                  │ name from counter/gauge used in the InstrumentHandler; turning off affected
│                  │ promhttp handlers; adding custom middleware before promhttp handler that
│                  │ will sanitize the request method given by Go http.Request; and using a
│                  │ reverse proxy or web application firewall, configured to only allow a
│                  │ limited set of methods.
├──────────────────┼───────────────────────────────────────────────────────────────
│ OSS Index ID     │ CVE-2022-21698
├──────────────────┼───────────────────────────────────────────────────────────────
│ CVSS Score       │ 7.5/10 (High)
├──────────────────┼───────────────────────────────────────────────────────────────
│ CVSS Vector      │ CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
├──────────────────┼───────────────────────────────────────────────────────────────
│ Link for more info │ https://ossindex.sonatype.org/vulnerability/CVE-2022-21698?component-type=golang
└──────────────────┴───────────────────────────────────────────────────────────────
pkg:golang/github.com/libp2p/go-buffer-pool@v0.0.2
1 known vulnerabilities affecting installed version
┌──────────────────────────────────────────────────────────────────────────────┐
│ 1 vulnerability found                                                        │
├──────────────────┬───────────────────────────────────────────────────────────┤
│ Description      │ 1 non-CVE vulnerability found. To see more details, please create a free │
│                  │ account at https://ossindex.sonatype.org/ and request for this information │
│                  │ using your registered account                                │
├──────────────────┼───────────────────────────────────────────────────────────┤
│ OSS Index ID     │ sonatype-2022-3945                                        │
├──────────────────┼───────────────────────────────────────────────────────────┤
│ CVSS Score       │ 5.3/10 (Medium)                                           │
├──────────────────┼───────────────────────────────────────────────────────────┤
│ CVSS Vector      │ CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L              │
├──────────────────┼───────────────────────────────────────────────────────────┤
│ Link for more info │ https://ossindex.sonatype.org/vulnerability/sonatype-2022-3945 │
└──────────────────┴───────────────────────────────────────────────────────────┘
```

THANK YOU FOR CHOOSING

// HALBORN