// HALBORN

Thorchain

Security Assessment

Prepared by: **Halborn**

Date of Engagement: **July 28th, 2021 - September 6th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/20/2021	Gokberk Gulgun
0.2	Document Updates	09/05/2021	Ferran Celades
0.3	Final Review	09/07/2021	Gabi Urrutia
1.0	Remediation Plan	11/24/2021	Gokberk Gulgun
1.1	Remediation Plan Review	12/06/2021	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
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

ThorChain engaged Halborn to conduct a security assessment on their **SDK**, **Bifrost** and **Router Smart Contract** repositories beginning on July 28th, 2021 and ending September 06th, 2021. Halborn was provided access to the source code of the application and the testing environment to conduct security testing using tools to scan, detect, validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

1.2 AUDIT SUMMARY

The team at Halborn was provided five weeks for the engagement and assigned three full time security engineers to audit the security of the smart contract. The security engineers are 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 to achieve the following:

- Ensure that Thornode, Thornode Bifrost component and Thorchain Router Contract functionalities are intended.
- Identify potential security issues with Thornode, Bifrost and Thorchain Router component.
- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

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 structures. 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:

SDK & BIFROST:

- Research into architecture and purpose.
- Static Analysis of security for scoped structures, and imported functions. (gosec, shadow, staticcheck, errcheck,semgrep)
- Manual Assessment for discovering security vulnerabilities on the codebase.
- Review of node and Bifrost functionalities.
- Dynamic Analysis.

SMART CONTRACT:

- Research into architecture and purpose.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)

- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Testnet deployment (Truffle, Ganache)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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.

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

1.4 SCOPE

Thornode Component : https://gitlab.com/thorchain/thornode

Commit ID: dd1cdb2c678a60eca17093825d315128798a80ad

Bifrost Component : https://gitlab.com/thorchain/thornode/-/tree/develop/bifrost

Commit ID: bcd4aed15647d0a42270e1c297fe41d1d034fb65

Router Smart Contract Audit : https://gitlab.com/thorchain/ethereum/eth-router/-/blob/master/contracts/THORChain_Router.sol

Commit ID: 592741e7545df2d285d0bb02b5a85d1d1d423044

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	5	11	17

EXECUTIVE OVERVIEW

LIKELIHOOD

IMPACT

	(HAL-02) (HAL-05)			
(HAL-20) (HAL-21) (HAL-27)	(HAL-06) (HAL-07)	(HAL-01) (HAL-03) (HAL-04)		
(HAL-29) (HAL-30) (HAL-33)	(HAL-08) (HAL-09) (HAL-10) (HAL-25) (HAL-26) (HAL-28)			
(HAL-11) (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17) (HAL-18) (HAL-19) (HAL-22) (HAL-23) (HAL-24) (HAL-31) (HAL-32)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SDK - HARDCODED SIGNER PASSWORD	Medium	FUTURE RELEASE
SDK - SENSITIVE INFORMATION IN ENVIRONMENT VARIABLES	Medium	RISK ACCEPTED
SDK - USE OF WEAK AES CRYPTOGRAPHIC PRIMITIVE	Medium	SOLVED
SDK - IMPROPER INPUT VALIDATION ON THE ENCRYPTION/DECRYPTION ROUTINE	Medium	FUTURE RELEASE
SDK - MISCONFIGURED POD SECURITY POLICIES	Medium	FUTURE RELEASE
SDK - SINGLE POINT OF FAILURE ON THE BNB SEED	Low	FUTURE RELEASE
SDK - LACK OF SENDER CHECK ON THE YGGDRASIL FUNDING PROGRESS	Low	NOT APPLICABLE
SDK - ASGARD VAULT IS NOT UPDATED DURING THE ROUTER CONTRACT UPGRADE	Low	NOT APPLICABLE
SDK - LACK OF FEE ERROR CHECK ON THE WITHDRAW PROGRESS	Low	SOLVED
SDK - MISSING EVENT HANDLER ON THE UNBOUNDING PROGRESS	Low	NOT APPLICABLE
SDK - REDUNDANT CODE ON THE REWARD CALCULATION	Informational	SOLVED
SDK - LACK OF NODE STATUS UPDATE ON THE BONDING PROGRESS	Informational	SOLVED
SDK - LACK OF MECHANISM ON THE THIRD PARTY CHAIN CLIENT UPGRADE	Informational	FUTURE RELEASE
SDK - MISSING STATUS DEFINITION ON THE BANNED NODE	Informational	NOT APPLICABLE
SDK - LACK OF CPU LIMIT DEFINITION ON THE POD CONFIGURATIONS	Informational	FUTURE RELEASE
SDK - REDUNDANT STATEMENT ON THE HALT CHECKS	Informational	NOT APPLICABLE
SDK - NETWORK CENTRALIZATION	Informational	ACKNOWLEDGED
SDK - HARDCODED ADMIN ADDRESSES IN THE GENESIS	Informational	ACKNOWLEDGED
SDK - HARDCODED MNEMONIC PHRASE IN THE REPOSITORY	Informational	NOT APPLICABLE

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
BIFROST - MISSING EVENT MEMO SIZE CHECK	Low	SOLVED
BIFROST - GO ROUTINE LEAK ON THE SIGNER	Low	SOLVED
BIFROST - REPETITIVE CODE	Informational	SOLVED
BIFROST - REDUNDANT CODE	Informational	SOLVED
BIFROST - CODE IMPROVEMENT ON THE TIME FUNCTION	Informational	SOLVED
SMART CONTRACT - INCOMPLETE EVENT	Low	RISK ACCEPTED
SMART CONTRACT - USE OF INSECURE TRANSFERS	Low	FUTURE RELEASE
SMART CONTRACT -IGNORED RETURN VALUES	Low	RISK ACCEPTED
SMART CONTRACT -LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR	Low	RISK ACCEPTED
SMART CONTRACT -MISSING AMOUNT CHECK	Low	RISK ACCEPTED
SMART CONTRACT -PRAGMA TOO RECENT	Low	FUTURE RELEASE
SMART CONTRACT - USAGE OF BLOCK-TIMESTAMP	Informational	ACKNOWLEDGED
SMART CONTRACT - FOR LOOP OVER DYNAMIC ARRAY	Informational	ACKNOWLEDGED
SMART CONTRACT - NO TEST COVERAGE	Informational	SOLVED

THORNODE COMPONENT SDK

3.1 (HAL-01) SDK - HARDCODED SIGNER PASSWORD - MEDIUM

Description:

Hardcoded Passwords are plain text passwords or other secrets in the source code. In the Thornode, SIGNERPASSWORD variable is hardcoded in the source code.

Code Location:

Bifrost Bash File

Listing 1: ethereum-block-scanner.go (Lines)

```
816 SIGNER_NAME="${SIGNER_NAME:=thorchain}"
817 SIGNER_PASSWD="${SIGNER_PASSWD:=password}"
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider deleting the hardcoded password from the source code. During the installation, signer-password should be taken from user input. This measure will protect the user from the additional attack vectors. Finally, the password requirements should be securely implemented.

Remediation Plan:

FUTURE RELEASE: ThorChain Team will fix the vulnerability on the future release.

3.2 (HAL-02) SDK - SENSITIVE INFORMATION IN ENVIRONMENT VARIABLES - MEDIUM

Description:

When the secret keys stored in an environment variable, It is prone to accidentally exposing them. Given that the environment is implicitly available to the process, it's hard, if not impossible, to track access and how the contents get exposed
(**ps -eww**)

It's common to have applications grab the whole environment and print it out for debugging or error reporting. Environment variables are passed down to child processes, which allows for unintended access. This breaks the principle of least privilege. Imagine that as part of your application, you call to a third-party tool to perform some action---all of a sudden that third-party tool has access to your environment. When applications crash, it's common for them to store the environment variables in log-files for later debugging.

Listing 2

```
Example Environment Variable

 - name: SIGNER_NAME
   value: thorchain
 - name: SIGNER_PASSWD
   value: password
```

Remediation:

It is recommended to store the **SIGNER_PASSWORD** in the secure Kubernetes storage.

Reference:

https://kubernetes.io/docs/concepts/configuration/secret/
https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/

Remediation Plan:

RISK ACCEPTED: The Thorchain Team decided to continue with the current environment variables.

3.3 (HAL-03) SDK - USE OF WEAK AES CRYPTOGRAPHIC PRIMITIVE - MEDIUM

Description:

Incorrect uses of encryption algorithms may result in sensitive data exposure, key leakage, broken authentication. Many cryptographic algorithms and protocols should not be used because they have been shown to have significant weaknesses or are otherwise insufficient for modern security requirements.
In Thornode, encryption via passphrase is completed through weak algorithm.

Code Location:

Encryption - Line #20

Listing 3

```
    func createHash(key string) (string, error) {
    hasher := md5.New()
    _, err := hasher.Write([]byte(key))
    return hex.EncodeToString(hasher.Sum(nil)), err
    }

    hash, err := createHash(passphrase)
    block, _ := aes.NewCipher([]byte(hash))
```

Recommendation:

Consider using **scrypt** library for key derivation via salt. The application should ensure the hash is generated by a cryptographically secure algorithm.

Remediation Plan:

SOLVED: The Thorchain Team fixed that on the current release.

Fix

3.4 (HAL-04) SDK - IMPROPER INPUT VALIDATION ON THE ENCRYPTION/DECRYPTION ROUTINE - MEDIUM

Description:

During the encryption/decryption phase of the key, passphrase and data size is not validated before cryptographic operations. The size of passphrase and data size should be validated according to size.

Code Location:

Encryption Go Line #41-42

Listing 4

```
1 func Encrypt(data []byte, passphrase string) ([]byte, error)
2 func Decrypt(data []byte, passphrase string) ([]byte, error)
```

Recommendation:

Input validation must be done to ensure only properly formed data is entering the encryption/decryption routine. Consider checking the size and type of the input.

Remediation Plan:

FUTURE RELEASE: ThorChain Team will fix the vulnerability on the future release.

3.5 (HAL-05) SDK - MISCONFIGURED POD SECURITY POLICIES - MEDIUM

Description:

A Pod Security Policy (PSP) is a cluster-wide policy that specifies security requirements/defaults for Pods to execute within the cluster. PSP allows us to create security mechanisms for Pods in Cluster. While security mechanisms are frequently specified within Pod/deployment configurations, PSPs define a minimum security threshold that all Pods must meet. PSPs are useful technical controls for enforcing cluster security measures. PSPs are especially useful for clusters managed by administrators with varying levels of authority. In these cases, top-level administrators can impose defaults to impose requirements on users.

Code Location:

Thornode Pod Security Context

Listing 5

```
serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the
      fullname template
  name:

priorityClassName: ""

podSecurityContext: {}
  # fsGroup: 2000

securityContext: {}
  # capabilities:
  #   drop:
  #   - ALL
  # readOnlyRootFilesystem: true
```

```
18   # runAsNonRoot: true
19   # runAsUser: 1000
```

Pod Security Context Items:

- privileged - Controls whether Pods can run privileged containers.
- hostPID, hostIPC - Controls whether containers can share host process namespaces.
- hostNetwork - Controls whether containers can use the host network.
- allowedHostPaths - Limits containers to specific paths of the host file system.
- readOnlyRootFilesystem - Requires the use of a read only root file system.
- runAsUser, runAsGroup, supplementalGroups, fsGroup - Controls whether container applications can run with root privileges or with root group membership.
- allowPrivilegeEscalation - Restricts escalation to root privileges.
- seLinux - Sets the SELinux context of the container.
- AppArmor annotations - Sets the AppArmor profile used by containers.
- seccomp annotations - Sets the seccomp profile used to sandbox containers.

Recommendation:

Before PSPs can be used, the PodSecurityPolicy plugin for the Kubernetes admission controller, which is part of kube-apiserver, must be enabled. In addition, the policy must be rightly approved using Role-Based Access Control (RBAC). Administrators should check the functionality of implemented PSPs from each role in their cluster's organization. Administrators should exercise caution in environments containing multiple PSPs, as Pod. Consider configuring all pod security policies for avoiding unexpected attack vectors.

Pod Security Policies

Remediation Plan:

FUTURE RELEASE: ThorChain Team will fix the vulnerability on the future release.

3.6 (HAL-06) SDK - SINGLE POINT OF FAILURE ON THE BNB SEED - LOW

Description:

In the **Thornode** status bash file, Thornode sync progress is completed according to connecting to **BNB seeds**. However, in the BNB seed only one peer is defined therefore if the peer is down, Thornode could not sync with **BNB** chain.

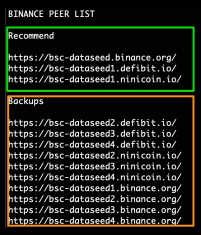
Code Location:

Thornode Bash File Line #45-46

Listing 6: BNB SYNC

```
1
2   [ "$NET" = "mainnet" ] && BNB_PEER=dataseed1.binance.org ||
        BNB_PEER=data-seed-pre-0-s1.binance.org
3   BNB_HEIGHT=$(curl -sL --fail -m 10 $BNB_PEER/status | jq -r ".
        result.sync_info.latest_block_height")
4   BNB_SYNC_HEIGHT=$(curl -sL --fail -m 10 binance-daemon:"
        $BINANCE_PORT"/status | jq -r ".result.sync_info.index_height
        ")
5   BNB_PROGRESS=$(calc_progress "$BNB_SYNC_HEIGHT" "$BNB_HEIGHT")
```

Binance Peer List:



Recommendation:

It is recommended to add minimum three peers into bash scripts. In these scripts, all peers should be checked if any peer is down.

Binance Peers

Remediation Plan:

FUTURE RELEASE: ThorChain Team will fix the vulnerability on the future release.

3.7 (HAL-07) SDK - LACK OF SENDER CHECK ON THE YGGDRASIL FUNDING PROGRESS - LOW

Description:

Input validation is performed to ensure only properly formed data is entering the workflow in a system. On the outbound handler, It has been observed that asgard vault has not been checked on the yggdrasil funding.

Code Location:

YGGDRASIL FUND

Listing 7: handler-observed-txin.go (Lines)

```
if vault.IsYggdrasil() && memo.IsType(TxYggdrasilFund) {
            // only add the fund to yggdrasil vault when the memo
               is yggdrasil+
            // no one should send fund to yggdrasil vault , if
               somehow scammer / airdrop send fund to yggdrasil
               vault
            // those will be ignored
            // also only asgard will send fund to yggdrasil , thus
                doesn't need to have confirmation counting
            if !voter.UpdatedVault {
                vault.AddFunds(tx.Tx.Coins)
                voter.UpdatedVault = true
            }
            vault.RemovePendingTxBlockHeights(memo.GetBlockHeight
               ())
}

```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to add transaction address check on the relevant conditions.

Pseudocode example below:

Listing 8: Rule Set (Lines)

```
1  tx.Tx.FromAddress(AsgardVaultAddress)
```

Remediation Plan:

NOT APPLICABLE: The Thorchain Team claims that the behaviour of the code is intended.

3.8 (HAL-08) SDK - ASGARD VAULT IS NOT UPDATED DURING THE ROUTER CONTRACT UPGRADE - LOW

Description:

During the router contract upgrade progress, on the upgradeContract function only yggdrasil vault is updated. Asgard vault's contract is not updated. That can cause a malfunction in the behaviour of asgard.

Code Location:

Handler Bond Go Line #173

Listing 9: router-upgrade-controller.go (Lines)

```
1 func (r *RouterUpgradeController) upgradeContract(ctx cosmos.
      Context) error {
2 ....
3         if vault.IsType(YggdrasilVault) {
4             // update yggdrasil vault to use new router contract
5             vault.UpdateContract(chainContract)
6         }
7         if err := r.mgr.Keeper().SetVault(ctx, vault); err != nil
              {
8             ctx.Logger().Error("fail to save vault", "error", err)
9         }
10
11     }
12
13 ....
```

YGGDrasil Manager Line #280

Listing 10

```
1 func (ymgr YggMgrV1) shouldFundYggdrasil(ctx cosmos.Context,
      asgard, ygg Vault, chain common.Chain) bool {
```

```
    asgardContract := asgard.GetContract(chain)
    if asgardContract.IsEmpty() {
        // the request chain doesn't support contract
        return true
    }
    yggContract := ygg.GetContract(chain)
    if asgardContract.Router.Equals(yggContract.Router) {
        return true
    }
    return false
}

```

Recommendation:

During the upgrade mechanism, asgard vault should be upgraded with the contract address of the router.

Remediation Plan:

NOT APPLICABLE: The Thorchain Team claims that the behaviour of the code is intended.

3.9 (HAL-09) SDK - LACK OF FEE ERROR CHECK ON THE WITHDRAW PROGRESS - LOW

During the withdrawal progress, if the **RUNE** amount is less than the transaction fee, the outbound transaction is cancelled. However, the error check has been deleted on the versioned withdraw handler.

Code Location:

https://gitlab.com/thorchain/thornode/-/blob/develop/x/thorchain/handler_withdraw.go

Listing 11: handler-withdraw.go (Lines)

```
1     if !runeAmt.IsZero() {
2         toi := TxOutItem{
3             Chain:     common.RuneAsset().GetChain(),
4             InHash:    msg.Tx.ID,
5             ToAddress: lp.RuneAddress,
6             Coin:      common.NewCoin(common.RuneAsset(), runeAmt)
                  ,
7             Memo:      memo,
8         }
9         // there is much much less chance thorchain doesn't have
              enough RUNE
10        okRune, err := h.mgr.TxOutStore().TryAddTxOutItem(ctx, h.
              mgr, toi)
11        if err != nil {
12            // when assetAmount is zero , the network didn't try
                  to send asset to customer
13            // thus if it failed to send RUNE , it should restore
                  pool here
14            // this might happen when the emit RUNE from the
                  withdraw is less than `NativeTransactionFee`
15            if assetAmount.IsZero() {
16                if err := h.mgr.Keeper().SetPool(ctx, pool); err
                      != nil {
```

```
17                      return nil, ErrInternal(err, "fail to save
                            pool")
18                  }
19              }
20              // emitted asset doesn't enough to cover fee, continue
21              return nil, multierror.Append(errFailAddOutboundTx,
                    err)
22          }
23
24          if !okRune {
25              return nil, errFailAddOutboundTx
26          }
27      }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider adding deleted error check on the related function.

Listing 12: handlerV1 code - Deleted Condition - ErrNotEnoughToPayFee

```
1       if !runeAmt.IsZero() {
2           toi := TxOutItem{
3               Chain:     common.RuneAsset().GetChain(),
4               InHash:    msg.Tx.ID,
5               ToAddress: lp.RuneAddress,
6               Coin:      common.NewCoin(common.RuneAsset(), runeAmt)
                    ,
7               Memo:      memo,
8           }
9           // there is much much less chance thorchain doesn't have
                enough RUNE
10          okRune, err := h.mgr.TxOutStore().TryAddTxOutItem(ctx, h.
                mgr, toi)
11          if err != nil {
12              // emitted asset doesn't enough to cover fee, continue
13              if !errors.Is(err, ErrNotEnoughToPayFee) {
```

```
14                  return nil, multierror.Append(errFailAddOutboundTx
                        , err)
15              }
16              okRune = true
17          }
18
19          if !okRune {
20              return nil, errFailAddOutboundTx
21          }
22      }
```

Remediation Plan:

SOLVED: The Thorchain Team fixed that on the current release.

Fix

3.10 (HAL-10) SDK - MISSING EVENT HANDLER ON THE UNBOUNDING PROGRESS - LOW

The unbonding progress does not emit the event after the progress. Events are a method of informing the transaction initiator about the actions taken by the called function. It logs its emitted parameters in a specific log history, which can be accessed outside the contract using some filter parameters.

Code Location:

https://gitlab.com/thorchain/thornode/-/blob/develop/x/thorchain/handler_unbond.go

Listing 13: Function (Lines)

```
1     if err := refundBond(ctx, msg.TxIn, msg.Amount, &na, h.mgr);
          err != nil {
2         return ErrInternal(err, "fail to unbond")
3     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider as much as possible declaring events at the end of function. Events can be used to detect the end of the operation.

Listing 14

```
    unBondEvent := NewEventUnbond(msg.Bond, BondPaid, msg.TxIn)
    if err := h.mgr.EventMgr().EmitEvent(ctx, bondEvent); err !=
        nil {
        return cosmos.Wrapf(errFailSaveEvent, "fail to emit bond
            event: %w", err)
    }
```

Remediation Plan:

NOT APPLICABLE: The Thorchain Team confirms that the event has been checked on the handler.

3.11 (HAL-11) SDK - REDUNDANT CODE ON THE REWARD CALCULATION - INFORMATIONAL

Description:

During the manual code review, it has been observed that there is a redundant code on the manager-network go file. **TotalReserve** and **TotalRewards** variables are not used.

Code Location:

Handler Bond Go Line #654-659

Listing 15: manager-network-v63.go (Lines)

```
1     // Move Rune from the Reserve to the Bond and Pool Rewards
2     totalRewards := bondReward.Add(totalPoolRewards)
3     if totalRewards.GT(totalReserve) {
4         totalRewards = totalReserve
5     }
6     totalReserve = common.SafeSub(totalReserve, totalRewards)
```

Recommendation:

Delete redundant code from the repository.

Remediation Plan:

SOLVED: The Thorchain Team fixed that on the current release.

Fix

3.12 (HAL-12) SDK - LACK OF NODE STATUS UPDATE ON THE BONDING PROGRESS - INFORMATIONAL

Description:

During the bonding progress, One Rune send to node account however if it fails, nodeacount status didnt set to **NodeUnknown**. NodeAccount status stays **NodeWhitelisted**.

Code Location:

Handler Bond Go Line #172

Listing 16: handler-bond.go (Lines)

```
1     if acct == nil && nodeAccount.Bond.GTE(cosmos.NewUint(common.
          One)) {
2         coin := common.NewCoin(common.RuneNative, cosmos.NewUint(
              common.One))
3         if err := h.mgr.Keeper().SendFromModuleToAccount(ctx,
              BondName, msg.NodeAddress, common.NewCoins(coin)); err
              != nil {
4             ctx.Logger().Error("fail to send one RUNE to node
                  address", "error", err)
5         }
6         nodeAccount.Bond = common.SafeSub(nodeAccount.Bond, cosmos
              .NewUint(common.One))
7         msg.Bond = common.SafeSub(msg.Bond, cosmos.NewUint(common.
              One))
8     }
```

Listing 17: handler-bond.go (Lines)

```
1     if nodeAccount.Status == NodeUnknown {
2         // white list the given bep address
3         nodeAccount = NewNodeAccount(msg.NodeAddress,
              NodeWhiteListed, emptyPubKeySet, "", cosmos.ZeroUint(),
               msg.BondAddress, common.BlockHeight(ctx))
```

```
4           ctx.EventManager().EmitEvent(
5               cosmos.NewEvent("new_node",
6                   cosmos.NewAttribute("address", msg.NodeAddress.
                        String()),
7               ))
8       }
```

Recommendation:

When a node is bonded for the first time , one RUNE is sent to the node address, however if it fails, the node status is not updated.

Remediation Plan:

SOLVED: The Thorchain Team fixed that on the current release.

Fix

3.13 (HAL-13) SDK - LACK OF MECHANISM ON THE THIRD PARTY CHAIN CLIENT UPGRADE - INFORMATIONAL

Description:

During the audit, the vulnerability has been found on the third-party chain (For instance : https://github.com/ethereum/go-ethereum/releases/tag/v1.10.8), It has been observed that third-party image is updated manually. However, this progress should be updated automatically.

Code Location:

GETH UPDATE DUE TO VULNERABILITY

Listing 18

```
version: '3'

services:
  ethereum-localnet:
    container_name: ethereum-localnet
    restart: unless-stopped
    image: ethereum/client-go:v1.10.8
    environment:
      ETH_BLOCK_TIME: ${ETH_BLOCK_TIME:-5}
    ports:
      - "8545:8545"
      - "30301:30301"
      - "30303:30303"
    volumes:
      - "../../scripts:/docker/scripts"
    entrypoint: "/docker/scripts/ethereum-mock.sh"
```

Recommendation:

It is recommended that the Kubernetes configuration files allow automatic updates.

Reference:

https://kubernetes.io/docs/concepts/containers/images/

Remediation Plan:

FUTURE RELEASE: The ThorChain Team will fix the vulnerability on the future release.

3.14 (HAL-14) SDK - MISSING STATUS DEFINITION ON THE BANNED NODE - INFORMATIONAL

Description:

In the **Thornode**, The multiple node status has been defined according to node attributes. However, on the banning progress, node ban status is depends on the two different variables. They are named as **ForcedToLeave** and **LeaveScore**. These variables have been checked on the every **TX-IN** and **TX-OUT** process.

Code Location:

Listing 19

```
/thornode-develop/x/thorchain/manager_slasher_v63.go
            }
            if toBan {
                na.ForcedToLeave = true
                na.LeaveScore = 1 // Set Leave Score to 1, which
                    means the nodes is bad
            }
        }
```

Recommendation:

It is recommended to add new node status type on the node account. Instead of checking two variables, the single status definition should create a more stable environment.

Remediation Plan:

NOT APPLICABLE: The Thorchain Team confirms that the banned nodes have NodeDisabled status.

3.15 (HAL-15) SDK - LACK OF CPU LIMIT DEFINITION ON THE POD CONFIGURATIONS - INFORMATIONAL

Description:

In Kubernetes, there are two resource policies that can be used for the namespace and node-level limitations. LimitRange allows limiting CPU and disk usages in any namespace pod and/or container. Only one LimitRange constraint can be created for each namespace. ResourceQuota, on the other hand, serves to constrain the total CPU / RAM usage on the namespace. During the manual code review, It has been observed that CPU LimitRange definition is missing on the node launcher.

Code Location:

Pod LimitRange Definition

Listing 20

```
resources:
  requests:
    cpu: 1
    memory: 2Gi
  limits:
    # cpu: 2
    memory: 4Gi

nodeSelector: {}

tolerations: []

affinity: {}

global: {}
```

Recommendation:

It is recommended to define limits on the pod configuration files.

Remediation Plan:

FUTURE RELEASE: The ThorChain Team will fix the vulnerability on the future release.

3.16 (HAL-16) SDK - REDUNDANT STATEMENT ON THE HALT CHECKS - INFORMATIONAL

Description:

When the network is halted, swap and liquidity operations are resulted with refund process. However, there is a redundant check which is applied on the **helpers** class.

Code Location:

Halt Checks

Listing 21

```
1     if isSwap || isAddLiquidity {
2         if isTradingHalt(ctx, m, h.mgr) || h.mgr.Keeper().
              RagnarokInProgress(ctx) {
3             if txIn.Tx.Coins.IsEmpty() {
4                 return &cosmos.Result{}, nil
5             }
6             if newErr := refundTx(ctx, txIn, h.mgr, h.mgr.
                  GetConstants(), se.ErrUnauthorized.ABCICode(), "
                  trading halted", targetModule); nil != newErr {
7                 return nil, ErrInternal(newErr, "trading is halted
                      , fail to refund")
8             }
9             return &cosmos.Result{}, nil
10        }
11    }
```

However, these checks can be completed with the calling **isTradingHaltV63** function. The redundant statement can be deleted from the condition checks.

Recommendation:

It is recommended to change the **if** condition statements to the **isTradingHaltV63** function.

Remediation Plan:

NOT APPLICABLE: The Thorchain Team confirms that the function is called by isTradingHalt().

3.17 (HAL-17) SDK - NETWORK CENTRALIZATION - INFORMATIONAL

Description:

Thorchain has stated that decentralization is an eventual goal, with a promise of transitioning to a DAO with full on-chain governance main-net. However, the chain has millions of dollars of assets are located in the chain In the some features, the implementations are designed by **MIMIRs**. Mimir is a feature to allow admins to change constants in the chain.

Code Location:

Listing 22: MIMIRS

```
IncentiveCurve
BlocksPerYear
OutboundTransactionFee
NativeTransactionFee
PoolCycle
MinRunePoolDepth
MaxAvailablePools
StagedPoolCost
MinimumNodesForYggdrasil
MinimumNodesForBFT
DesiredValidatorSet
AsgardSize
ChurnInterval
ChurnRetryInterval
ValidatorsChangeWindow
LeaveProcessPerBlockHeight
BadValidatorRedline
BadValidatorRate
OldValidatorRate
LowBondValidatorRate
LackOfObservationPenalty
SigningTransactionPeriod
DoubleSignMaxAge
MinimumBondInRune
FundMigrationInterval
```

```
26      ArtificialRagnarokBlockHeight
27      MaximumLiquidityRune
28      StrictBondLiquidityRatio
29      DefaultPoolStatus
30      FailKeygenSlashPoints
31      FailKeysignSlashPoints
32      LiquidityLockUpBlocks
33      ObserveSlashPoints
34      ObservationDelayFlexibility
35      YggFundLimit
36      YggFundRetry
37      JailTimeKeygen
38      JailTimeKeysign
39      NodePauseChainBlocks
40      MinSwapsPerBlock
41      MaxSwapsPerBlock
42      MaxSynthPerAssetDepth
43      VirtualMultSynths
44      MinSlashPointsForBadValidator
45      FullImpLossProtectionBlocks
46      BondLockupPeriod
47      NumberOfNewNodesPerChurn
48      MinTxOutVolumeThreshold
49      TxOutDelayRate
50      TxOutDelayMax
51      MaxTxOutOffset
52      TNSRegisterFee
53      TNSFeeOnSale
54      TNSFeePerBlock
55      PermittedSolvencyGap
56
```

Recommendation:

There are multiple critical functionalities that depend on the MIMIRs. MIMIRs should be disabled on the mainnet.

Remediation Plan:

ACKNOWLEDGED: The Thorchain Team claims that MIMIR is a requirement to run the network safely for the time being.

3.18 (HAL-18) SDK - HARDCODED ADMIN ADDRESSES IN THE GENESIS - INFORMATIONAL

Description:

The Admin accounts are defined at genesis and there does not seem to be a mechanism to update the account. If the key is compromised it would be impossible to contain the attack without stopping the chain and deploying new code. This further amplifies the risk of mimir updates because there is no containment mechanism in the case of a compromised key, and the only resulting possible solution is to halt network.

Code Location:

Listing 23: "x/thorchain/mimir-address.go

```
1 package thorchain
2
3 // ADMINS hard coded admin address
4 var ADMINS = []string{
5     "thor1xghvhe4p50aqh5zq2t2vls938as0dkr2mzgpgh",
6     "thor19pkncem64gajdwrd5kasspyj0t75hhkpqjn5qh",
7 }
```

Recommendation:

Consider to eliminate the Admin role and use governance.

Remediation Plan:

ACKNOWLEDGED: The Thorchain Team claims that these addresses are hard-coded, but the intention is to remove them eventually.

3.19 (HAL-19) SDK - HARDCODED MNEMONIC PHRASE IN THE REPOSITORY - INFORMATIONAL

Description:

In the ThorNode repository, it was discovered that there is a mnemonic phrase hardcoded in test script.

Code Location:

ed25519_keys_test.go

Listing 24: ed25519-keys-test.go (Lines)

```
1     mnemonic := `grape safe sound obtain bachelor festival profit
          iron meat moon exit garbage chapter promote noble grocery
          blood letter junk click mesh arm shop decorate`
```

Testnet Address

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Use secure vault to store credentials instead of using them hardcoded. Also, It is recommended to delete mnemonic phrases from the Git history, and change any previously used mnemonic phrases to prevent future incidents if these mnemonic phrases have been leaked, or reused elsewhere. Test scripts should dynamically create keys at the runtime.

Remediation Plan:

NOT APPLICABLE: The Thorchain Team claims that these addresses are just used for the testnet.

3.20 SDK MANUAL TESTING

ABUSING MATHEMATICAL CALCULATIONS:

Description:

During the these tests, liquidity management calculation are examined. The issues are investigated which can lead to integer overflow. The software which performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.

Code Location:

Listing 25: handler-add-liquidity.go (Lines 379)

```
1 func calculatePoolUnitsV1(oldPoolUnits, poolRune, poolAsset,
      addRune, addAsset cosmos.Uint) (cosmos.Uint, cosmos.Uint, error
      ) {
2     if addRune.Add(poolRune).IsZero() {
3         return cosmos.ZeroUint(), cosmos.ZeroUint(), errors.New("
              total RUNE in the pool is zero")
4     }
5     if addAsset.Add(poolAsset).IsZero() {
6         return cosmos.ZeroUint(), cosmos.ZeroUint(), errors.New("
              total asset in the pool is zero")
7     }
8     if poolRune.IsZero() || poolAsset.IsZero() {
9         return addRune, addRune, nil
10    }
11    P := cosmos.NewDecFromBigInt(oldPoolUnits.BigInt())
12    R := cosmos.NewDecFromBigInt(poolRune.BigInt())
13    A := cosmos.NewDecFromBigInt(poolAsset.BigInt())
14    r := cosmos.NewDecFromBigInt(addRune.BigInt())
15    a := cosmos.NewDecFromBigInt(addAsset.BigInt())
```

However, all amounts are stored into **Uint**. The overflow has not been found on the code base. Performing multiplication before division can sometimes avoid loss of precision. In the mathematical calculations,

divide operation has been completed before multiply.

TXIN AND TXOUT OBSERVATIONS:

Description:

During the observation of inbound and output transaction, the checks are checked. During inbound transaction, the transaction should be signed and cannot direct to outbound. These conditions have been checked on the following code base.

Listing 26: thornode-develop/x/thorchain/types/msg-observed-txin.go

```
func (m *MsgObservedTxIn) ValidateBasic() error {
    if m.Signer.Empty() {
        return cosmos.ErrInvalidAddress(m.Signer.String())
    }
    if len(m.Txs) == 0 {
        return cosmos.ErrUnknownRequest("Txs cannot be empty")
    }
    for _, tx := range m.Txs {
        if err := tx.Valid(); err != nil {
            return cosmos.ErrUnknownRequest(err.Error())
        }
        obAddr, err := tx.ObservedPubKey.GetAddress(tx.Tx.Coins
            [0].Asset.GetChain())
        if err != nil {
            return cosmos.ErrUnknownRequest(err.Error())
        }
        if !tx.Tx.ToAddress.Equals(obAddr) {
            return cosmos.ErrUnknownRequest("request is not an
                inbound observed transaction")
        }
        if len(tx.Signers) > 0 {
            return cosmos.ErrUnknownRequest("signers must be empty
                ")
        }
        if len(tx.OutHashes) > 0 {
            return cosmos.ErrUnknownRequest("out hashes must be
                empty")
        }
        if tx.Status != Status_incomplete {
            return cosmos.ErrUnknownRequest("status must be
                incomplete")
        }
    }
```

```
29
30      return nil
31  }
```

Listing 27: thornode-develop/x/thorchain/types/msg-observed-txin.go

```
1  func (m *MsgObservedTxOut) ValidateBasic() error {
2      if m.Signer.Empty() {
3          return cosmos.ErrInvalidAddress(m.Signer.String())
4      }
5      if len(m.Txs) == 0 {
6          return cosmos.ErrUnknownRequest("Txs cannot be empty")
7      }
8      for _, tx := range m.Txs {
9          if err := tx.Valid(); err != nil {
10             return cosmos.ErrUnknownRequest(err.Error())
11         }
12         obAddr, err := tx.ObservedPubKey.GetAddress(tx.Tx.Coins
               [0].Asset.GetChain())
13         if err != nil {
14             return cosmos.ErrUnknownRequest(err.Error())
15         }
16         if !tx.Tx.FromAddress.Equals(obAddr) {
17             return cosmos.ErrUnknownRequest("Request is not an
                   outbound observed transaction")
18         }
19         if len(tx.Signers) > 0 {
20             return cosmos.ErrUnknownRequest("signers must be empty
                   ")
21         }
22         if len(tx.OutHashes) > 0 {
23             return cosmos.ErrUnknownRequest("out hashes must be
                   empty")
24         }
25         if tx.Status != Status_incomplete {
26             return cosmos.ErrUnknownRequest("status must be
                   incomplete")
27         }
28     }
```

```
29      return nil
30  }
```

UNBOUNDED WITHDRAW TESTS:

Description:

During the tests, Thornode's withdraw handler have been checked. The withdraw amount has been checked on the multiple code sections.

Code Location:

Listing 28: thornode-develop/x/thorchain/types/msg-withdraw-liquidity.go (Lines 379)

```
func (m *MsgWithdrawLiquidity) ValidateBasic() error {
    if m.Signer.Empty() {
        return cosmos.ErrInvalidAddress(m.Signer.String())
    }
    // here we can't call m.Tx.Valid , because we allow user to send withdraw request without any coins in it
    // m.Tx.Valid will reject this kind request , which result withdraw to fail
    if m.Tx.ID.IsEmpty() {
        return cosmos.ErrInvalidAddress("tx id cannot be empty")
    }
    if m.Asset.IsEmpty() {
        return cosmos.ErrUnknownRequest("pool asset cannot be empty")
    }
    if m.Asset.IsRune() {
        return cosmos.ErrUnknownRequest("asset cannot be rune")
    }
    if m.WithdrawAddress.IsEmpty() {
        return cosmos.ErrUnknownRequest("address cannot be empty")
    }
    if m.BasisPoints.IsZero() {
        return cosmos.ErrUnknownRequest("basis points can't be zero")
    }
    if m.BasisPoints.GT(cosmos.NewUint(MaxWithdrawBasisPoints)) {
        return cosmos.ErrUnknownRequest("basis points is larger than maximum withdraw basis points")
    }
    if !m.WithdrawalAsset.IsEmpty() && !m.WithdrawalAsset.IsRune()
        && !m.WithdrawalAsset.Equals(m.Asset) {
```

```
26          return cosmos.ErrUnknownRequest("withdrawal asset must be
                empty, rune, or pool asset")
27      }
28      return nil
29  }
```

EXAMINATION OF TRANSACTION TYPES:

Description:

In the Thornode, there are multiple type of transactions are defined. Thornode is working with memos. Therefore, every behaviour is designed according to memos. The following memos are pre-defined in the repository.

Listing 29

```
1 TxAdd, TxWithdraw, TxSwap, TxDonate, TxBond, TxUnbond, TxLeave,
      TxSwitch, TxReserve, TxNoOp, TxTHORName,TxYggdrasilFund,
      TxYggdrasilReturn, TxMigrate, TxConsolidate,TxOutbound,
      TxRefund, TxRagnarok
```

According to memo types, the destination of transaction have been checked.

Listing 30: Inbound Transactions

```
1 TxAdd, TxWithdraw, TxSwap, TxDonate, TxBond, TxUnbond, TxLeave,
      TxSwitch, TxReserve, TxNoOp, TxTHORName
```

Listing 31: Outbound Transactions

```
1 TxOutbound, TxRefund, TxRagnarok
```

Listing 32: Internal Transactions

```
1 TxYggdrasilFund, TxYggdrasilReturn, TxMigrate, TxConsolidate
```

After the tests, the following behaviours are observed.

Listing 33

```
1 Deposit Handler allows -> Inbound Transactions
2 ObservedIn Transactions -> Inbound Transactions
3 ObservedOut Transactions -> Outbound and Internal Transactions
```

According to Thornode documentations, these implementations have been

completed via handler codes.

CONSENSUS BASED MESSAGE TYPE ANALYSIS:

Description:

In the **ThorNode**, the handlers are used for the checking multiple operations on the consensus. Some operations do not have consensus mechanism. Therefore, during this test, the functionalities are examined. The following handlers are dependent on the consensus.

Listing 34: Consensus Based Handlers

```
    m[MsgTssPool{}.Type()] = NewTssHandler(mgr)
    m[MsgObservedTxIn{}.Type()] = NewObservedTxInHandler(mgr)
    m[MsgObservedTxOut{}.Type()] = NewObservedTxOutHandler(mgr)
    m[MsgTssKeysignFail{}.Type()] = NewTssKeysignHandler(mgr)
    m[MsgErrataTx{}.Type()] = NewErrataTxHandler(mgr)
    m[MsgMimir{}.Type()] = NewMimirHandler(mgr)
    m[MsgBan{}.Type()] = NewBanHandler(mgr)
    m[MsgNetworkFee{}.Type()] = NewNetworkFeeHandler(mgr)
```

Listing 35: Non-Consensus Node Client Handlers

```
    m[MsgSetNodeKeys{}.Type()] = NewSetNodeKeysHandler(mgr)
    m[MsgSetVersion{}.Type()] = NewVersionHandler(mgr)
    m[MsgSetIPAddress{}.Type()] = NewIPAddressHandler(mgr)
```

Listing 36: Non-Consensus Native Handlers

```
    m[MsgSend{}.Type()] = NewSendHandler(mgr)
    m[MsgDeposit{}.Type()] = NewDepositHandler(mgr)
    m[MsgSolvency{}.Type()] = NewSolvencyHandler(mgr)
```

As a result of this test, all consensus mechanisms are implemented according to documentation.

3.21 SDK STATIC ANALYSIS REPORT

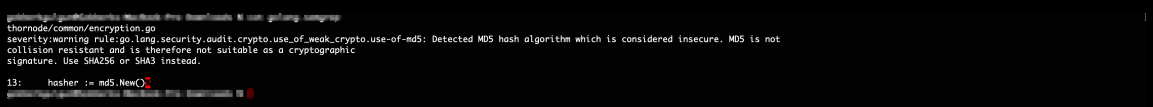
SDK STATIC ANALYSIS:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped component. Among the tools used were staticcheck, gosec ineffassign and others. 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.

Semgrep - Security Analysis Output Sample:

Listing 37: Rule Set (Lines)

```
1 semgrep --config "p/halborn-go" thornode --exclude='*_test.go' -o
      halborn.semgrep
```



```
1: ... Function(fn),args)};defer(counter);return counter};clearTask=function clearImmediate(id){delete queue[id]};if(__webpack_require__(61)(process)=="process"){defer ...
[Shortened a long line from output, adjust with --max-chars-per-line]
----------------------------------------------------------------------------
1: ... nction(fn),args)};defer(counter);return counter};clearTask=function clearImmediate(id){delete queue[id]};if(__webpack_require__(56)(process)=="process"){defer=f ...
[Shortened a long line from output, adjust with --max-chars-per-line]
----------------------------------------------------------------------------
8: ... ction isFunction(object){return Object.prototype.toString.call(object)==="[object Function]"}module.exports=new Type("tag:yaml.org,2002:js/function",{kind:"scal ...
[Shortened a long line from output, adjust with --max-chars-per-line]
----------------------------------------------------------------------------
8: ... (object)==="[object Function]"]module.exports=new Type("tag:yaml.org,2002:js/function",{kind:"scalar",resolve:resolveJavascriptFunction,construct:constructJavas ...
[Shortened a long line from output, adjust with --max-chars-per-line]
----------------------------------------------------------------------------
severity:warning rule:typescript.react.security.audit.react-dangerouslysetinnerhtml.react-dangerouslysetinnerhtml: Setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site script
ing (XSS) attack.

1: ... sName:{0,_classnames2.default)(className,"markdown"),dangerouslySetInnerHTML:{__html:sanitized}}}exports.default=Markdown;function sanitizer(str){return _dompu ...
[Shortened a long line from output, adjust with --max-chars-per-line]
----------------------------------------------------------------------------
1: ... uslySetInnerHTML:{__html:trimmed},className:(0,_classnames2.default)(className,"renderedMarkdown")}})return null};exports.default=(0,_helpers.OAS3ComponentWrapF ...
[Shortened a long line from output, adjust with --max-chars-per-line]

thor-master/builtin/gen/bindata.go
severity:warning rule:go.lang.security.decompression_bomb.potential-dos-via-decompression-bomb: Detected a possible denial-of-service via a zip bomb attack. By limiting the max bytes read, you can mitigate this attack. `io
.CopyN()` can specify a size. Refer to https://bomb.codes/ to learn more about this attack and other ways to mitigate it.
autofix: s/(.*)(Copy|CopyBuffer)\((.*?),(.*?)(\)|,.*\))/\1CopyN(\3, \4, 1024*1024*256)/g
54:     _, err = io.Copy(&buf, gz)

thor-master/logdb/logdb.go
severity:warning rule:go.lang.security.audit.database.string-formatted-query.string-formatted-query: String content [illegible] detected in [illegible] could lead to SQL injection if
the string is not sanitized properly. Audit this call to ensure the
SQL is not manipulatable by external data.

46:     if _, err := db.Exec(configTableSchema + refTableScheme + eventTableSchema + transferTableSchema); err != nil {

thor-master/p2psrv/discv5/ticket.go
severity:warning rule:go.lang.security.audit.crypto.math_random.math-random-used: Do not use `math/rand`. Use `crypto/rand` instead.
764:                    val = rand.Int()

thor-master/tracers/internal/tracers/assets.go
severity:warning rule:go.lang.security.decompression_bomb.potential-dos-via-decompression-bomb: Detected a possible denial-of-service via a zip bomb attack. By limiting the max bytes read, you can mitigate this attack. `io
.CopyN()` can specify a size. Refer to https://bomb.codes/ to learn more about this attack and other ways to mitigate it.
autofix: s/(.*)(Copy|CopyBuffer)\((.*?),(.*?)(\)|,.*\))/\1CopyN(\3, \4, 1024*1024*256)/g
33:     _, err = io.Copy(&buf, gz)

thor-master/tracers/tracer.go
severity:warning rule:go.lang.security.audit.unsafe.use-of-unsafe-block: Using the unsafe package in Go gives you low-level memory management and
many of the strengths of the C language but also gives flexibility to the attacker
of your application.

52:     return *(*[]byte)(unsafe.Pointer(&sl))

thor-master/txpool/tx_pool.go
severity:warning rule:go.lang.security.audit.crypto.math_random.math-random-used: Do not use `math/rand`. Use `crypto/rand` instead.
185:            delay := time.Second * time.Duration(rand.Int()%60+60)

thornode/bifrost/observer/observe.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
118:    for {
119:            select {
120:            case <-o.stopChan:
121:                    o.sendDeck()
122:                    return
123:            case <-time.After(constants.ThorchainBlockTime):
124:                    o.sendDeck()
125:            }
126:    }

thornode/bifrost/pkg/chainclients/bitcoin/client.go
severity:warning rule:dgryski.semgrep-go.contexttodo.context-todo: Consider to use well-defined context
489:                    ctx := context.TODO()

thornode/bifrost/pkg/chainclients/bitcoincash/client.go
severity:warning rule:dgryski.semgrep-go.contexttodo.context-todo: Consider to use well-defined context
486:                    ctx := context.TODO()

thornode/bifrost/pkg/chainclients/dogecoin/client.go
severity:warning rule:dgryski.semgrep-go.contexttodo.context-todo: Consider to use well-defined context
485:                    ctx := context.TODO()

thornode/bifrost/pkg/chainclients/ethereum/unstuck.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
35:     for {
36:             select {
37:             case <-c.stopchan:
38:                     // time to exit
39:                     return
40:             case <-time.After(constants.ThorchainBlockTime):
41:                     c.unstuckAction()
42:             }
43:     }

thornode/bifrost/pkg/chainclients/litecoin/client.go
severity:warning rule:dgryski.semgrep-go.contexttodo.context-todo: Consider to use well-defined context
487:                    ctx := context.TODO()

thornode/bifrost/pubkeymanager/pubkey_manager.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
274:    for {
275:            select {
276:            case <-pkm.stopChan:
277:                    return
278:            case <-time.After(constants.ThorchainBlockTime):
279:                    pkm.fetchPubKeys()
280:            }
281:    }
```


// HALBORN

```
thornode/bifrost/tss/tss_signer.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
206:    for {
207:            select {
208:            case <-s.done:
209:                    // requested to exit
210:                    return
211:            case t := <-s.taskQueue:
212:                    taskLock.Lock()
213:                    _, ok := tasks[t.PoolPubKey]
214:                    if !ok {
215:                            tasks[t.PoolPubKey] = []*tssKeySignTask{
216:                                    t,
217:                            }
218:                    } else {
219:                            tasks[t.PoolPubKey] = append(tasks[t.PoolPubKey], t)
220:                    }
221:                    taskLock.Unlock()
222:            case <-time.After(time.Second):
223:                    // This implementation will check the tasks every second , and send whatever is in the queue to TSS
224:                    // if it has more than maxKeysignPerRequest(15) in the queue , it will only send the first maxKeysignPerRequest(15) of them
225:                    // the reset will be send in the next request
226:                    taskLock.Lock()
227:                    for k, v := range tasks {
228:                            if len(v) == 0 {
229:                                    delete(tasks, k)
230:                                    continue
231:                            }
232:                            totalTasks := len(v)
233:                            // send no more than maxKeysignPerRequest messages in a single TSS keysign request
234:                            if totalTasks > maxKeysignPerRequest {
235:                                    totalTasks = maxKeysignPerRequest
236:                                    // when there are more than maxKeysignPerRequest messages in the task queue need to be signed
237:                                    // the messages has to be sorted , because the order of messages that get into the slice is not deterministic
238:                                    // so it need to sorted to make sure all bifrosts send the same messages to tss
239:                                    sort.SliceStable(v, func(i, j int) bool {
240:                                            return v[i].Msg < v[j].Msg
241:                                    })
242:                            }
243:                            s.wg.Add(1)
244:                            signingTask := v[:totalTasks]
245:                            tasks[k] = v[totalTasks:]
246:                            go s.toLocalTSSSigner(k, signingTask)
247:                    }
248:                    taskLock.Unlock()
249:            }
250:    }

thornode/common/duration.go
severity:error rule:dgryski.semgrep-go.errnilcheck.err-nil-check: superfluous nil err check before return
33:             if err != nil {
34:                     return err
35:             }
36:             return nil
```


```
224:                    // if it has more than maxKeysignPerRequest(15) in the queue , it will only send the first maxKeysignPerRequest(15) of them
225:                    // the reset will be send in the next request
226:                    taskLock.Lock()
227:                    for k, v := range tasks {
228:                            if len(v) == 0 {
229:                                    delete(tasks, k)
230:                                    continue
231:                            }
232:                            totalTasks := len(v)
233:                            // send no more than maxKeysignPerRequest messages in a single TSS keysign request
234:                            if totalTasks > maxKeysignPerRequest {
235:                                    totalTasks = maxKeysignPerRequest
236:                                    // when there are more than maxKeysignPerRequest messages in the task queue need to be signed
237:                                    // the messages has to be sorted , because the order of messages that get into the slice is not deterministic
238:                                    // so it need to sorted to make sure all bifrosts send the same messages to tss
239:                                    sort.SliceStable(v, func(i, j int) bool {
240:                                            return v[i].Msg < v[j].Msg
241:                                    })
242:                            }
243:                            s.wg.Add(1)
244:                            signingTask := v[:totalTasks]
245:                            tasks[k] = v[totalTasks:]
246:                            go s.toLocalTSSSigner(k, signingTask)
247:                    }
248:                    taskLock.Unlock()
249:            }
250:    }

thornode/common/duration.go
severity:error rule:dgryski.semgrep-go.errnilcheck.err-nil-check: superfluous nil err check before return
33:             if err != nil {
34:                     return err
35:             }
36:             return nil
```


```
thornode/x/thorchain/keeper/v1/keeper_liquidity_provider.go
severity:error rule:dgryski.semgrep-go.errnilcheck.err-nil-check: superfluous nil err check before return
68:     if err != nil {
69:             return record, err
70:     }
71:
72:     return record, nil

thornode/x/thorchain/manager_event_v1.go
severity:error rule:dgryski.semgrep-go.oddifsequence.odd-sequence-ifs: Odd sequence of ifs
56:     if feeEvent.Fee.Coins.IsEmpty() && feeEvent.Fee.PoolDeduct.IsZero() {
57:             return nil
58:     }
59:
60:     if feeEvent.Fee.Coins.IsEmpty() && feeEvent.Fee.PoolDeduct.IsZero() {
61:             return nil
62:     }

thornode/x/thorchain/query/query.go
severity:error rule:dgryski.semgrep-go.joinpath.use-strings-join-path: did you want path.Join() or filepath.Join()?
31:     return fmt.Sprintf("custom/%s", strings.Join(args, "/"))
```

Gosec - Security Analysis Output Sample:

```
/x/thorchain/types/test_common.go:145] - G404 (CWE-338): Use of weak random number generator (math/rand instead of crypto/rand) (Confidence: MEDIUM, Severity: HIGH)
  144: func GetRandomPubKey() common.PubKey {
> 145:         r := rand.New(rand.NewSource(time.Now().UnixNano()))
  146:         accts := simtypes.RandomAccounts(r, 1)


/x/thorchain/types/test_common.go:75] - G404 (CWE-338): Use of weak random number generator (math/rand instead of crypto/rand) (Confidence: MEDIUM, Severity: HIGH)
  74: func GetRandomBech32ConsensusPubKey() string {
> 75:         r := rand.New(rand.NewSource(time.Now().UnixNano()))
  76:         accts := simtypes.RandomAccounts(r, 1)


/x/thorchain/types/test_common.go:29] - G404 (CWE-338): Use of weak random number generator (math/rand instead of crypto/rand) (Confidence: MEDIUM, Severity: HIGH)
  28:         s := rand.NewSource(time.Now().UnixNano())
> 29:         r := rand.New(s)
  30:         accts := simtypes.RandomAccounts(r, 1)


/common/random_string.go:15] - G404 (CWE-338): Use of weak random number generator (math/rand instead of crypto/rand) (Confidence: MEDIUM, Severity: HIGH)
  14:         for i := 0; i < n; {
> 15:                 if idx := int(rand.Int63() & letterIdxMask); idx < len(letterBytes) {
  16:                         b[i] = letterBytes[idx]


/bifrost/pkg/chainclients/ethereum/tokens_db.go:16] - G101 (CWE-798): Potential hardcoded credentials (Confidence: LOW, Severity: HIGH)
  15:         // prefixTokenMeta declares prefix to use in leveldb to avoid conflict
> 16:         prefixTokenMeta = `eth-tokenmeta-`
  17: )


/common/encryption.go:13] - G401 (CWE-326): Use of weak cryptographic primitive (Confidence: HIGH, Severity: MEDIUM)
  12: func createHash(key string) (string, error) {
> 13:         hasher := md5.New()
  14:         _, err := hasher.Write([]byte(key))


/x/thorchain/types/test_common.go:182] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)
  181:         dir := path.Join(path.Dir(filename), "../../..")
> 182:         dat, err := ioutil.ReadFile(path.Join(dir, "version"))
  183:         if err != nil {


cmd/bifrost/main.go:268] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)
  262:         }
> 263:         buf, err := ioutil.ReadFile(file)
  264:         if err != nil {
```

```
thornode-develop/bifrost/pkg/chainclients/bitcoin/client.go:757] - G601 (CWE-118): Implicit memory aliasing in for loop. (Confidence: MEDIUM, Severity: MEDIUM)
  756:                 c.removeFromMemPoolCache(tx.Hash)
> 757:                 txInItem, err := c.getTxIn(&tx, block.Height)
  758:                 if err != nil {


thornode-develop/common/encryption.go:6] - G501 (CWE-327): Blocklisted import crypto/md5: weak cryptographic primitive (Confidence: HIGH, Severity: MEDIUM)
  5:  "crypto/cipher"
> 6:  "crypto/md5"
  7:  "crypto/rand"


/x/thorchain/manager_validator_v58.go:870] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
  869:                 }
> 870:                 iterator.Close()
  871:                 if count >= maxWithdrawsPerBlock {


/x/thorchain/manager_validator_v56.go:863] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
  862:                 }
> 863:                 iterator.Close()
  864:                 if count >= maxWithdrawsPerBlock {


/x/thorchain/manager_validator_v51.go:863] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
  862:                 }
> 863:                 iterator.Close()
  864:                 if count >= maxWithdrawsPerBlock {


thornode-develop/x/thorchain/manager_validator_v1.go:856] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
  855:                 }
> 856:                 iterator.Close()
  857:                 if count >= maxWithdrawsPerBlock {


thornode-develop/cmd/thornode/cmd/root.go:101] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
  100:                 // options accordingly.
> 101:                 serverCtx.Viper.BindPFlags(cmd.Flags())
  102:                 _, err := server.GetPruningOptionsFromFlags(serverCtx.Viper)


/thornode-develop/cmd/thornode/cmd/ed25519_keys.go:45] - G104 (CWE-703): Errors unhandled. (Confidence: HIGH, Severity: LOW)
  44:         cmd.PersistentFlags().String(flags.FlagKeyringBackend, flags.DefaultKeyringBackend, "Select keyring's backend (os|file|test)")
> 45:         viper.BindPFlag(flags.FlagKeyringBackend, cmd.Flags().Lookup(flags.FlagKeyringBackend))
  46:         return cmd
```

Staticcheck - Security Analysis Output Sample:

```
common/coin.go:66:10: error strings should not be capitalized (ST1005)
common/coin.go:69:10: error strings should not be capitalized (ST1005)
common/pubkey_test.go:32:2: field addrTHOR is unused (U1000)
x/thorchain/errors.go:45:2: error var notAuthorized should have name of the form errFoo (ST1012)
x/thorchain/genesis.go:375:45: should use make(map[string]int64) instead (S1019)
x/thorchain/handler_add_liquidity_test.go:23:2: field failGetNextEventID is unused (U1000)
x/thorchain/handler_ban_test.go:130:35: should use make(map[string]int64) instead (S1019)
x/thorchain/handler_leave_test.go:376:69: argument err is overwritten before first use (SA4009)
        x/thorchain/handler_leave_test.go:378:5: assignment to err
x/thorchain/handler_observed_txin_test.go:177:2: should use 'return addr.Equals(k.nas[0].NodeAddress)' instead of 'if addr.Equals(k.nas[0].NodeAddress) { return true }; return false' (S1008)
x/thorchain/handler_observed_txin_test.go:216:2: this value of err is never used (SA4006)
x/thorchain/handler_outbound_tx_test.go:415:2: this value of err is never used (SA4006)
x/thorchain/handler_outbound_tx_test.go:452:2: this value of err is never used (SA4006)
x/thorchain/handler_reserve_contrib_test.go:22:2: field errGetReserveContributors is unused (U1000)
x/thorchain/handler_reserve_contrib_test.go:23:2: field errSetReserveContributors is unused (U1000)
x/thorchain/handler_solvency_test.go:72:2: this value of err is never used (SA4006)
x/thorchain/handler_solvency_test.go:110:2: this value of err is never used (SA4006)
x/thorchain/handler_swap.go:174:11: error strings should not be capitalized (ST1005)
x/thorchain/handler_swap.go:242:11: error strings should not be capitalized (ST1005)
x/thorchain/handler_swap.go:260:22: func SwapHandler.validateV63 is unused (U1000)
x/thorchain/handler_swap.go:264:22: func SwapHandler.validateCurrent is unused (U1000)
x/thorchain/handler_swap.go:314:11: error strings should not be capitalized (ST1005)
x/thorchain/handler_swap_test.go:55:2: field hasEvent is unused (U1000)
x/thorchain/handler_swap_test.go:114:39: should use make(map[common.Asset]Pool) instead (S1019)
x/thorchain/handler_swap_test.go:213:2: this value of err is never used (SA4006)
x/thorchain/handler_swap_test.go:280:2: this value of err is never used (SA4006)
x/thorchain/handler_swap_test.go:307:2: this value of err is never used (SA4006)
x/thorchain/handler_switch_test.go:28:2: this value of destination is never used (SA4006)
x/thorchain/handler_test.go:33:5: error var kaboom should have name of the form errFoo (ST1012)
x/thorchain/handler_test.go:33:14: error strings should not end with punctuation or a newline (ST1005)
x/thorchain/handler_test.go:66:2: var multiPerm is unused (U1000)
x/thorchain/handler_test.go:67:2: var randomPerm is unused (U1000)
x/thorchain/handler_test.go:68:2: var holder is unused (U1000)
x/thorchain/handler_test.go:326:34: unnecessary use of fmt.Sprintf (S1039)
x/thorchain/handler_test.go:492:34: unnecessary use of fmt.Sprintf (S1039)
x/thorchain/handler_tss_keysign_test.go:27:2: field members is unused (U1000)
x/thorchain/handler_tss_test.go:31:2: field keygenBlock is unused (U1000)
x/thorchain/handler_unbond_test.go:19:5: error var returnYggErr should have name of the form errFoo (ST1012)
x/thorchain/handler_unbond_test.go:78:2: should use 'return k.vault.PubKey.Equals(pkey)' instead of 'if k.vault.PubKey.Equals(pkey) { return true }; return false' (S1008)
x/thorchain/handler_withdraw.go:988:35: func WithdrawLiquidityHandler.handleV63 is unused (U1000)
x/thorchain/handler_withdraw.go:992:35: func WithdrawLiquidityHandler.handleCurrent is unused (U1000)
x/thorchain/handler_yggdrasil_test.go:76:2: field pool is unused (U1000)
x/thorchain/helpers.go:1086:2: empty branch (SA9003)
x/thorchain/helpers_test.go:297:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:302:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:307:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:348:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:374:2: field errSetEvent is unused (U1000)
x/thorchain/helpers_test.go:582:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:589:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:590:2: this value of err is never used (SA4006)
x/thorchain/helpers_test.go:621:2: this value of err is never used (SA4006)
x/thorchain/keeper/keeper_dummy.go:19:5: error var kaboom should have name of the form errFoo (ST1012)
x/thorchain/keeper/keeper_dummy.go:19:14: error strings should not end with punctuation or a newline (ST1005)
x/thorchain/manager_slasher_v48_test.go:290:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v48_test.go:306:35: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v54_test.go:106:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v54_test.go:204:37: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v54_test.go:261:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v54_test.go:290:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v54_test.go:306:35: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v58_test.go:106:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v58_test.go:204:37: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v58_test.go:261:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v58_test.go:290:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v58_test.go:306:35: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v63_test.go:106:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v63_test.go:204:37: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v63_test.go:261:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v63_test.go:290:36: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_slasher_v63_test.go:306:35: should use make(map[string]int64) instead (S1019)
x/thorchain/manager_swap_queue_v1.go:121:39: should use make(map[common.Asset]Pool) instead (S1019)
x/thorchain/manager_swap_queue_v47.go:112:39: should use make(map[common.Asset]Pool) instead (S1019)
x/thorchain/manager_swap_queue_v58.go:112:39: should use make(map[common.Asset]Pool) instead (S1019)
x/thorchain/manager_txout_v53_test.go:148:2: this value of err is never used (SA4006)
x/thorchain/manager_txout_v54_test.go:148:2: this value of err is never used (SA4006)
x/thorchain/manager_txout_v55_test.go:148:2: this value of err is never used (SA4006)
x/thorchain/manager_txout_v58_test.go:148:2: this value of err is never used (SA4006)
x/thorchain/manager_yggdrasil_v1.go:280:2: should use 'return asgardContract.Router.Equals(yggContract.Router)' instead of 'if asgardContract.Router.Equals(yggContract.Router) { return true }; return false' (S1008)
x/thorchain/manager_yggdrasil_v45.go:284:2: should use 'return asgardContract.Router.Equals(yggContract.Router)' instead of 'if asgardContract.Router.Equals(yggContract.Router) { return true }; return false' (S1008)
x/thorchain/manager_yggdrasil_v59.go:294:2: should use 'return asgardContract.Router.Equals(yggContract.Router)' instead of 'if asgardContract.Router.Equals(yggContract.Router) { return true }; return false' (S1008)
x/thorchain/manager_yggdrasil_v63.go:298:2: should use 'return asgardContract.Router.Equals(yggContract.Router)' instead of 'if asgardContract.Router.Equals(yggContract.Router) { return true }; return false' (S1008)
x/thorchain/memo/memo_switch.go:31:2: this value of destination is never used (SA4006)
x/thorchain/memo/memo_test.go:55:2: this value of memo is never used (SA4006)
x/thorchain/memo/memo_test.go:210:2: this value of memo is never used (SA4006)
x/thorchain/memo/memo_test.go:217:2: this value of memo is never used (SA4006)
x/thorchain/querier.go:1116:35: should use make(map[string]int64) instead (S1019)
x/thorchain/swap_v1_test.go:31:2: should use 'return !asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"})' instead of 'if asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"}) { return false }; return true' (S1008)
x/thorchain/types/msg_observed_txin_test.go:48:2: this value of err is never used (SA4006)
x/thorchain/types/msg_observed_txout_test.go:48:2: this value of err is never used (SA4006)
x/thorchain/types/querier.go:9:2: package "gitlab.com/thorchain/thornode/common" is being imported more than once (ST1019)
        x/thorchain/types/querier.go:10:2: other import of "gitlab.com/thorchain/thornode/common"
x/thorchain/types/type_ban.go:71:2: should use 'return HasSuperMajority(count, len(nodeAccounts))' instead of 'if HasSuperMajority(count, len(nodeAccounts)) { return true }; return false' (S1008)
x/thorchain/types/type_errata.go:58:2: should use 'return HasSuperMajority(count, len(nas))' instead of 'if HasSuperMajority(count, len(nas)) { return true }; return false' (S1008)
x/thorchain/types/type_observed_network_fee_voter.go:57:2: should use 'return HasSuperMajority(count, len(nas))' instead of 'if HasSuperMajority(count, len(nas)) { return true }; return false' (S1008)
x/thorchain/types/type_observed_tx.go:140:2: should use 'return len(m.OutHashes) >= numOuts' instead of 'if len(m.OutHashes) >= numOuts { return true }; return false' (S1008)
x/thorchain/types/type_solvency_voter.go:60:2: should use 'return HasMinority(count, len(nas))' instead of 'if HasMinority(count, len(nas)) { return true }; return false' (S1008)
x/thorchain/types/type_tss.go:83:43: should use make(map[common.Chain]int) instead (S1019)
x/thorchain/types/type_tss_metric_test.go:28:2: this value of median is never used (SA4006)
x/thorchain/types/type_tx_out.go:17:10: error strings should not be capitalized (ST1005)
x/thorchain/types/type_tx_out.go:20:10: error strings should not be capitalized (ST1005)
x/thorchain/withdraw_v1_test.go:39:2: should use 'return !asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"})' instead of 'if asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"}) { return false }; return true' (S1008)
x/thorchain/withdraw_v55_test.go:36:2: should use 'return !asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"})' instead of 'if asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"}) { return false }; return true' (S1008)
x/thorchain/withdraw_v58_test.go:36:2: should use 'return !asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"})' instead of 'if asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"}) { return false }; return true' (S1008)
x/thorchain/withdraw_v63_test.go:36:2: should use 'return !asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"})' instead of 'if asset.Equals(common.Asset{Chain: common.BNBChain, Symbol: "NOTEXIST", Ticker: "NOTEXIST"}) { return false }; return true' (S1008)
```


HALBORN

Ineffassign - Security Analysis Output Sample:

According to the test results, some of the findings found by these tools were considered as false positives while some of these findings were real security concerns. All relevant findings were reviewed by the auditors and relevant findings addressed in the report as security concerns.

BIFROST

4.1 (HAL-20) BIFROST - MISSING EVENT MEMO SIZE CHECK - LOW

Description:

Input validation is performed to ensure only properly formed data is entering the workflow in a system. On the Ethereum block scanner, It has been observed that event memo is not validated even if it is empty. When software does not validate input properly, an attacker can craft the input that is not expected by the component.

Code Location:

ethereum_block_scanner.go Line #816

Listing 38: ethereum-block-scanner.go (Lines)

```
816     if len(txInItem.Memo) > 0 && !strings.EqualFold(txInItem.Memo,
            depositEvt.Memo)
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Both event and transaction item memos size should be checked in the application. The input validation should be performed on the memo. This progress can be completed by a regex check.

Remediation plan:

SOLVED: Issue fixed in commit Merge Request - 1920.

4.2 (HAL-21) BIFROST - GO ROUTINE LEAK ON THE SIGNER - LOW

Description:

Concurrency features like Goroutines and channels are built into the language and runtime to reduce or eliminate the need for libraries. Go routine leaking is a type of memory leak. In the bifrost signer component, there is potential go routine leak due to unbuffered channel.

Code Location:

signer/sign.go Line #180

Listing 39: signer.go (Lines)

```
180 func (s *Signer) runWithContext(ctx context.Context, fn func()
        error) error {
181     ch := make(chan error)
182     go func() {
183         ch <- fn()
184     }()
185     select {
186     case <-ctx.Done():
187         return ctx.Err()
188     case err := <-ch:
189         return err
190     }
191 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider to add signal into make function.

Listing 40

```
1 func (s *Signer) runWithContext(ctx context.Context, fn func()
      error) error {
2     ch := make(chan error,1)
3     go func() {
4         ch <- fn()
5     }()
6     select {
7     case <-ctx.Done():
8         return ctx.Err()
9     case err := <-ch:
10        return err
11    }
12 }
```

Remediation plan:

SOLVED: Issue fixed in commit Merge Request - 1852.

4.3 (HAL-22) BIFROST - REPETITIVE CODE - INFORMATIONAL

Description:

In the Ethereum block scanner, events are categorized according to router and smart contract address. However, the code is repeated on the related code part. The check could be completed with a single function.

Code Location:

ethereum_block_scanner.go Line #686

Listing 41: ethereum-block-scanner.go (Lines)

```
686 func (e *ETHScanner) isTHORChainRouterAddress(addr ecommon.Address
      ) bool {
687     // get the smart contract used by thornode
688     contractAddresses := e.pubkeyMgr.GetContracts(common.ETHChain)
689     for _, item := range contractAddresses {
690         if strings.EqualFold(item.String(), addr.String()) {
691             return true
692         }
693     }
694     return false
695 }
696
697 func (e *ETHScanner) isToValidContractAddress(addr *ecommon.
      Address) bool {
698     // get the smart contract used by thornode
699     contractAddresses := e.pubkeyMgr.GetContracts(common.ETHChain)
700     // combine the whitelist smart contract address
701     for _, item := range append(contractAddresses,
          whitelistSmartContractAddres...) {
702         if strings.EqualFold(item.String(), addr.String()) {
703             return true
704         }
705     }
706     return false
707 }
```

```
708
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to delete repetitive code from the repository.

Remediation plan:

SOLVED: Issue fixed in commit Merge Request - 1852.

4.4 (HAL-23) BIFROST - REDUNDANT CODE - INFORMATIONAL

Description:

(*Observer).isOutboundMsg function has not been used in the bifrost component.

Code Location:

observer/observe.go Line #254

Listing 42: observer/observe.go (Lines)

```
254 func (o *Observer) isOutboundMsg(chain common.Chain, fromAddr
       string) bool {
255     matchOutbound, _ := o.pubkeyMgr.IsValidPoolAddress(fromAddr,
           chain)
256     if matchOutbound {
257         return true
258     }
259     return false
260 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to delete redundant code from the repository.

Remediation plan:

SOLVED: Issue fixed in commit Merge Request - 1852.

4.5 (HAL-24) BIFROST - CODE IMPROVEMENT ON THE TIME FUNCTION - INFORMATIONAL

Description:

The `time.Since` helper has the same effect as using `time.Now().Sub(x)` but is easier to read.

Code Location:

BlockScanner.go Line#162
Bitcoin/Client.go Line#254
BitcoinCash/Client.go Line#251
Ethereum.go Line#777
Litecoin/Client.go Line#2562

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use `time.Since` instead of `time.Now().Sub(x)`.

Remediation plan:

SOLVED: Issue fixed in commit Merge Request - 1852.

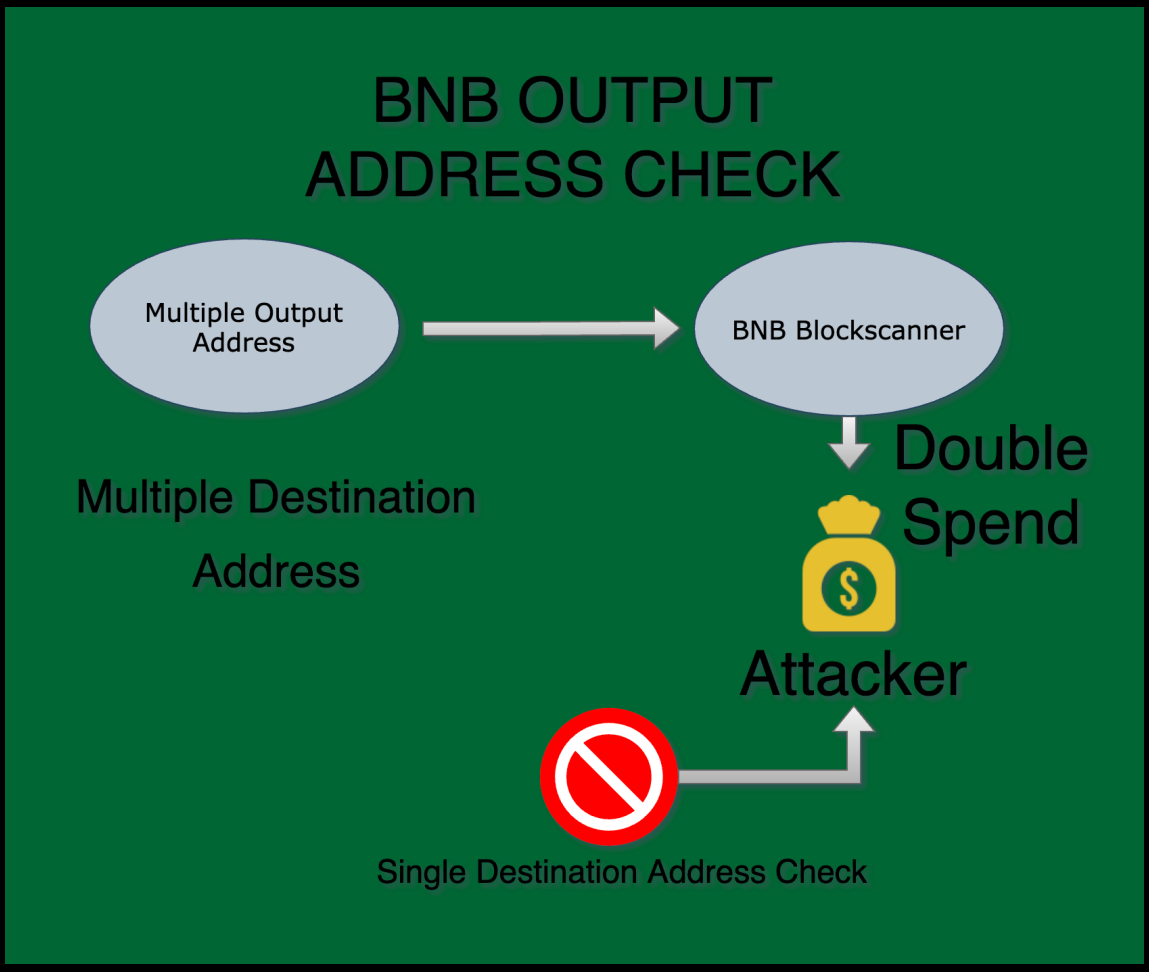
4.6 BIFROST MANUAL TESTING

BNB - BIFROST COMPONENT TESTING:

RECEIVER ADDRESS CHECK:

Description:

Binance blockscanner use `outputs` array for parsing all other coins. The `output` array have been used for intercepting destination address. During this test, `output` array size have been checked for multiple destination addresses.



Code Location:

Listing 43: binance-block-scanner.go (Lines 379)

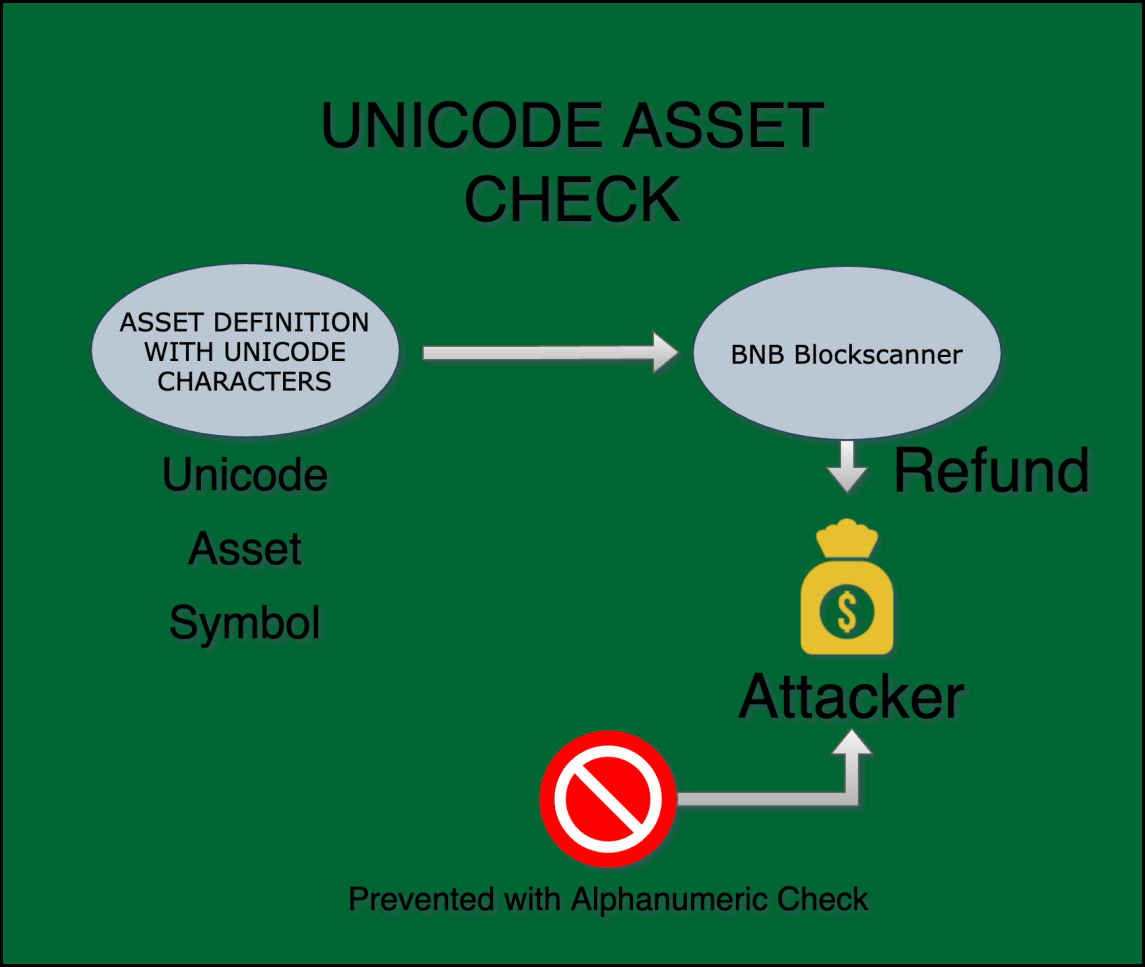
```
376 func (b *BinanceBlockScanner) getCoinsForTxIn(outputs []bmsg.
      Output, receiver string) (common.Coins, error) {
377     cc := common.Coins{}
378     for _, output := range outputs {
379         if receiver != output.Address.String() {
380             continue
381         }
382         for _, c := range output.Coins {
383             asset, err := common.NewAsset(fmt.Sprintf("BNB.%s", c.
                  Denom))
384             if err != nil {
385                 return nil, fmt.Errorf("fail to create asset, %s
                      is not valid: %w", c.Denom, err)
386             }
387
388             // skip mini tokens
389             if asset.Symbol.IsMiniToken() {
390                 continue
391             }
392             amt := cosmos.NewUint(uint64(c.Amount))
393             cc = append(cc, common.NewCoin(asset, amt))
394         }
395     }
396     return cc, nil
397 }
```

However, the receiver has been checked on the line 379.

UNICODE ASSET SYMBOL CHECK:

Description:

As a previous attack, symbol parsing section is directly attacked by hackers. Therefore, Halborn Team decided to check unicode assets name. Firstly, BNB side has been controlled on the definition of asset symbol. When we are observing the asset symbol definition the following text observed.



Listing 44

```
1 Symbol-string : The length of the string for representing this
      asset is between 3 and 8 alphanumeric characters and is case-
```

```
        insensitive. "B" suffixed symbol is also allowed for pegging
      to those tokens already exist on other chains. The symbol is
      suffixed with the first 3 bytes of the issue transaction hash
      to remove a constraint of requiring unique token names. The
      native token, BNB, does not require this suffix.
```

Reference

Although, the name section can be defined with unicode characters, symbol can be used only with alphanumeric characters. Therefore, BNB side is not vulnerable for fake asset symbols.

On the other hand, Asset symbol parse has been examined on the Thorchain side. However, only alphanumeric asset names are parsed. The code can be seen from the below.

Listing 45: common/symbol.go (Lines 25)

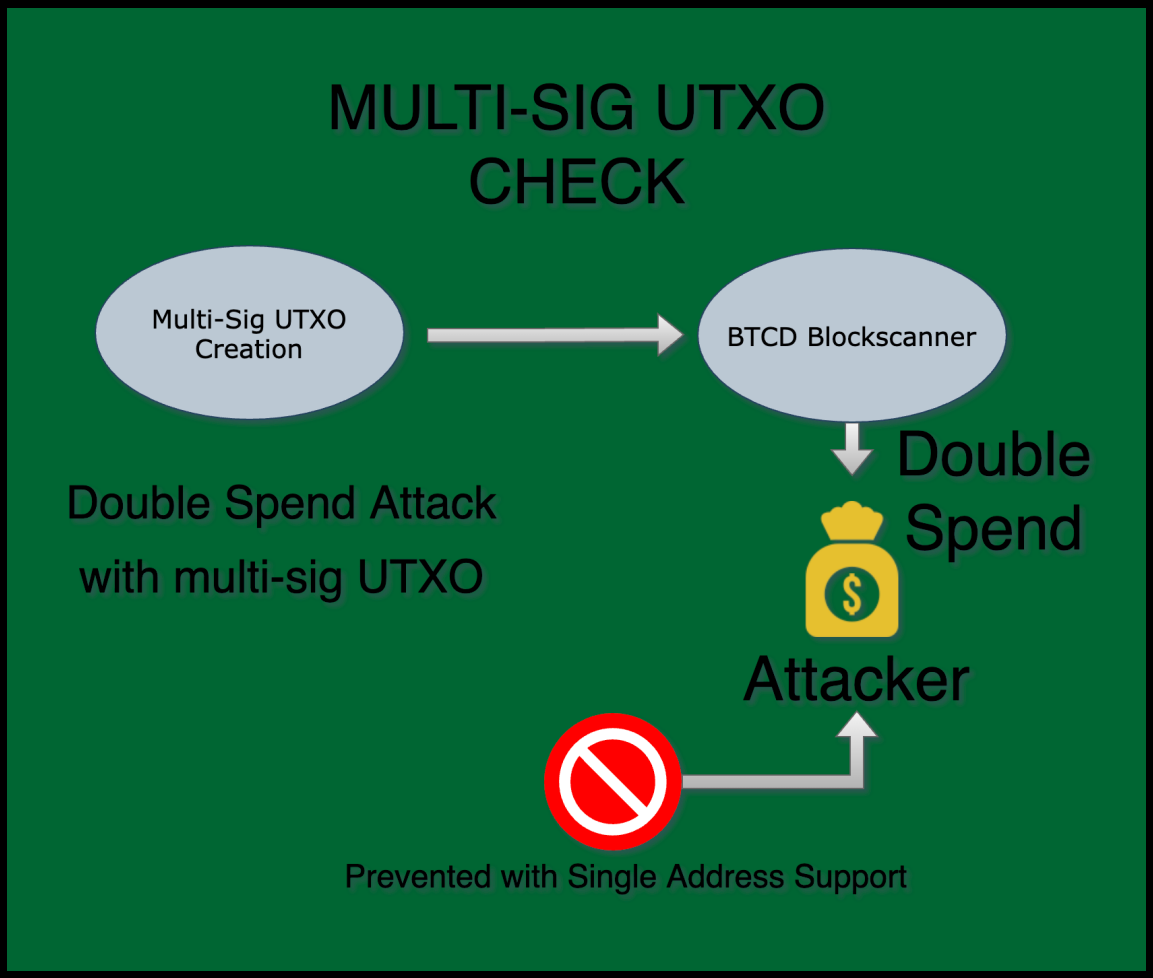
```
24 func NewSymbol(input string) (Symbol, error) {
25     if !isAlphaNumeric(input) {
26         return "", errors.New("invalid symbol")
27     }
28
29     return Symbol(strings.ToUpper(input)), nil
30 }
```

4.7 BTC/BCH/LTC - BIFROST COMPONENT TESTING

CUSTOM UTXO TESTING:

Description:

A UTXO defines an output of a blockchain transaction that has not been spent, i.e. can be used as an input in a new transaction. During the testing other chains, Double spend has been checked with custom UTXOs. Double-spending is a potential flaw in a digital cash scheme in which the same single digital token can be spent more than once. During the test phase, double spending has been checked on the bifrost components.



Code Location:

Listing 46: bitcoin/client.go (Lines 844)

```
839 func (c *Client) getOutput(sender string, tx *btcjson.TxRawResult,
        consolidate bool) (btcjson.Vout, error) {
840     for _, vout := range tx.Vout {
841         if strings.EqualFold(vout.ScriptPubKey.Type, "nulldata") {
842             continue
843         }
844         if len(vout.ScriptPubKey.Addresses) != 1 {
845             return btcjson.Vout{}, fmt.Errorf("no vout address
                    available")
846         }
847         if vout.Value > 0 {
848             if consolidate && vout.ScriptPubKey.Addresses[0] ==
                    sender {
849                 return vout, nil
850             }
851             if !consolidate && vout.ScriptPubKey.Addresses[0] !=
                    sender {
852                 return vout, nil
853             }
854         }
855     }
856     return btcjson.Vout{}, btypes.FailOutputMatchCriteria
857 }
```

Listing 47: bitcoin/client.go (Lines 817)

```
810 ...
811 for idx, vout := range tx.Vout {
812         if vout.Value > 0 {
813             countWithOutput++
814         }
815         // check we have one address on the first 2 outputs
816         // TODO check what we do if get multiple addresses
817         if idx < 2 && vout.ScriptPubKey.Type != "nulldata" && len(
                vout.ScriptPubKey.Addresses) != 1 {
818             return true
819         }
```

```
820     }
821     // none of the output has any value
822     if countWithOutput == 0 {
823         return true
824     }
825     // there are more than two output with value in it, not
            THORChain format
826     if countWithOutput > 2 {
827         return true
828     }
829     return false
830 ...
```

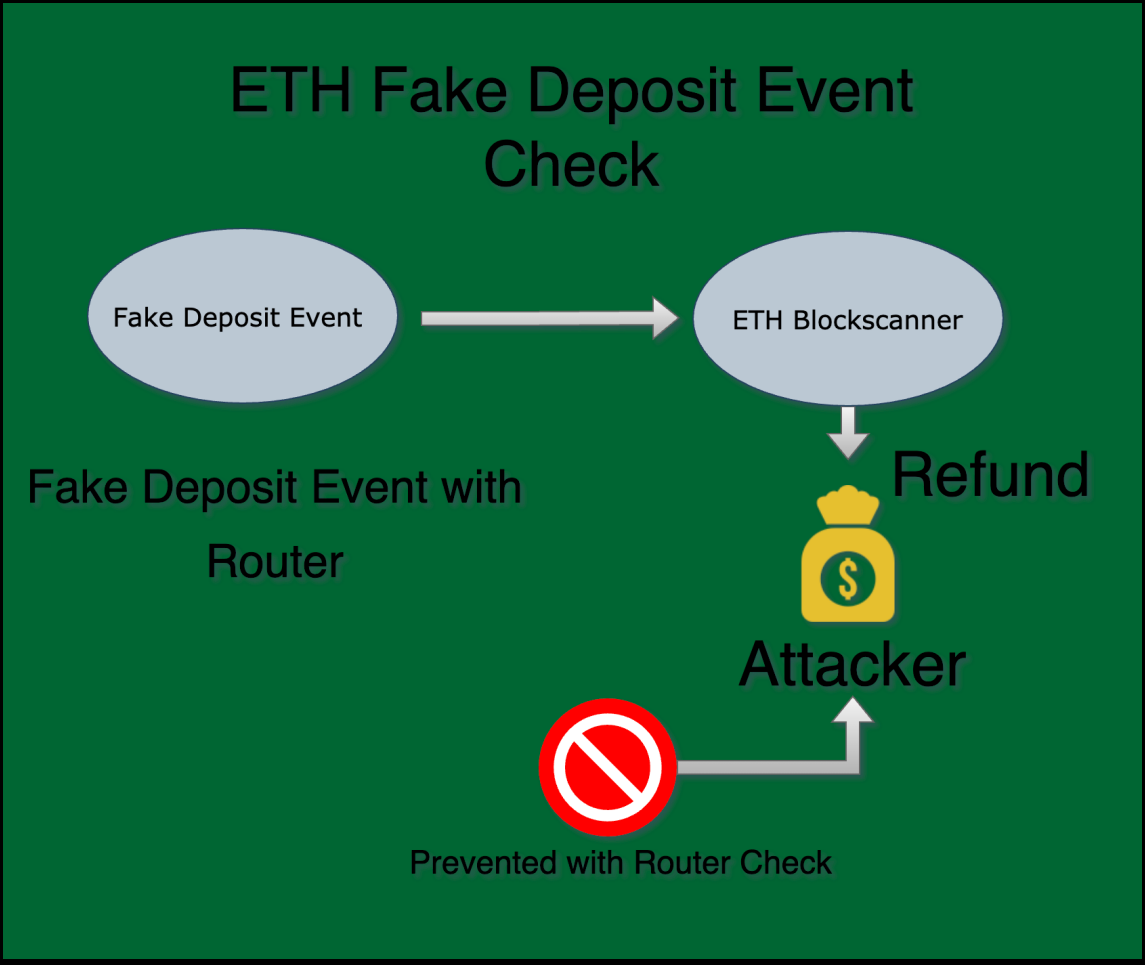
However, in the BTC,BCH,LTC the multiple output addresses has not been supported yet. As a conclusion, components are not vulnerable to double spend attacks.

4.8 ETH - BIFROST COMPONENT TESTING

FAKE DEPOSIT EVENT CHECK:

Description:

After the previous hack, `isTHORChainRouterAddress` control has been added into Ethereum block scanner. The applied control examined during the fake deposit testing.



Code Location:

Listing 48: ethereum-block-scanner.go (Lines 686)

```
686 func (e *ETHScanner) isTHORChainRouterAddress(addr ecommon.Address
    ) bool {
687     // get the smart contract used by thornode
688     contractAddresses := e.pubkeyMgr.GetContracts(common.ETHChain)
689     for _, item := range contractAddresses {
690         if strings.EqualFold(item.String(), addr.String()) {
691             return true
692         }
693     }
694     return false
695 }
```

With that protection mechanism, Thorchain fake deposit events have been prevented at the transaction for loop.

Listing 49: ethereum-block-scanner.go (Lines 806)

```
803     ...
804     for _, item := range receipt.Logs {
805         // only events produced by THORChain router is processed
806         if !e.isTHORChainRouterAddress(item.Address) {
807             continue
808         }
809     ...
```

To improving code readability, the duplicate for loops are implemented over the code structure. This is issue handled on the ‘‘.

FAKING OUT A REFUND:

Description:

During the test, Halborn Team tried to replay and manipulate memos for manipulating refund progress.

```
>>> router.transferOut("0x05355D84e86E9baB0D18d0479B3a37e70450d4f3", "0x0000000000000000000000000000000000000000", 492800000000000000
, "REFUND:05B7E655DA06155028554BD31B0D6E458E6F7AFF347D19160100A6A1AC9F57E0", {'from':accounts[0]})
Transaction sent: 0xac74cbe2f45f5fbf52aea0670052d64503a6bba3c5604163840579856102285b
  Gas price: 1.902993424 gwei   Gas limit: 40649   Nonce: 106
Waiting for confirmation...
```

Code Location:

Faking out a REFUND (replaying previous refund):

First Transaction : https://ropsten.etherscan.io/tx/0x5069FF66F8AF7595BDCDE63C930057

Second Transaction: https://ropsten.etherscan.io/tx/0xac74cbe2f45f5fbf52aea0670052d6

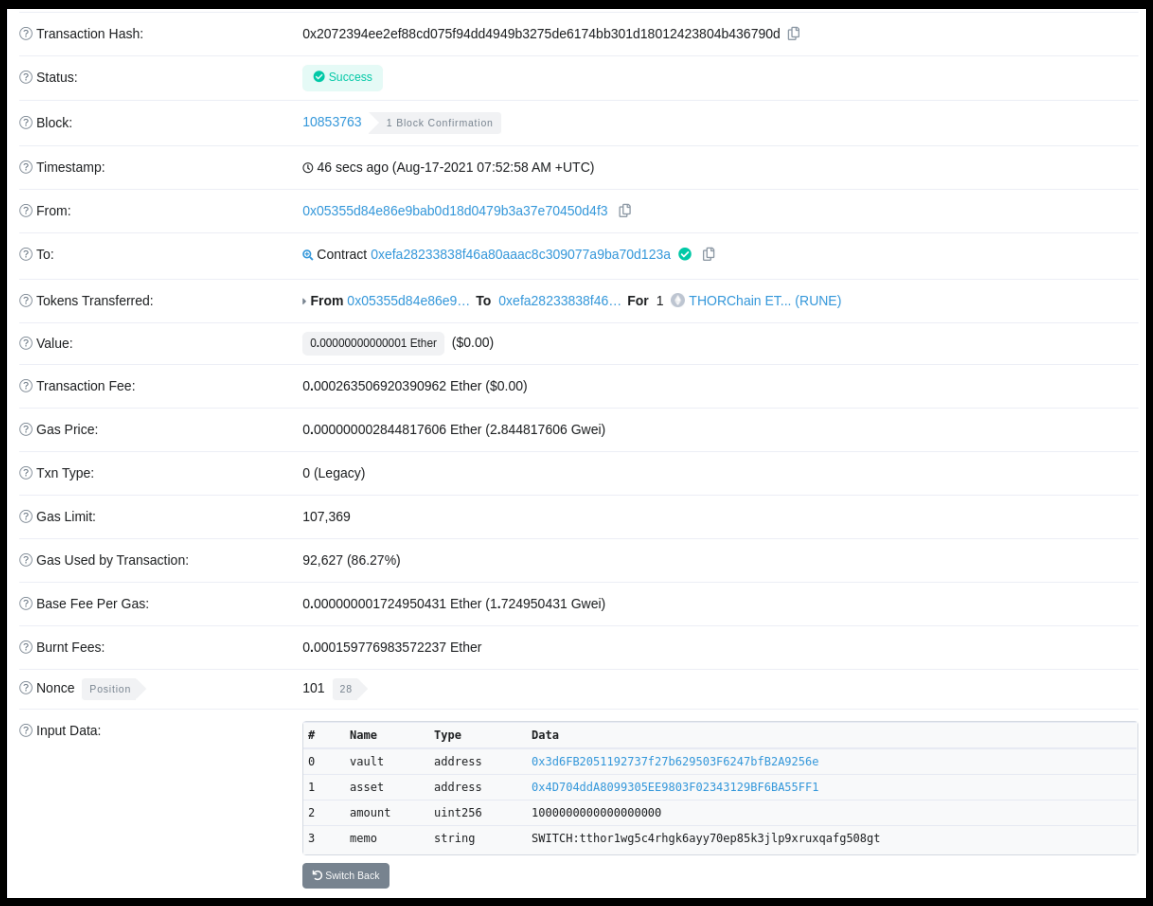
The manipulation tests have been failed due to these protection mechanisms. The protection mechanism placed on the memos.

FAKING OUT AN ASSET NAME:

Description:

During the test, Halborn Team tried to create a fake asset name and symbol as a ThorChain Rune.

Fake Token Generation



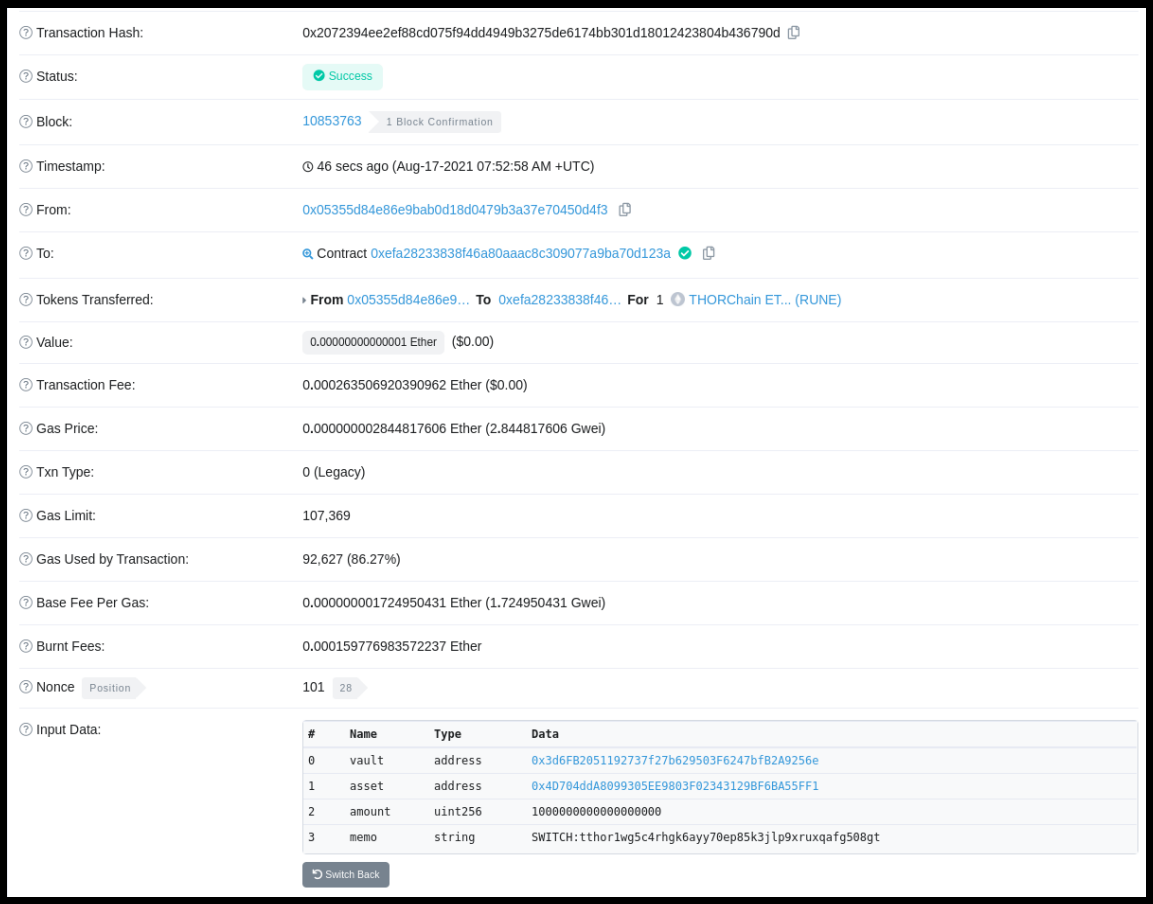
Transaction Hash:	0x2072394ee2ef88cd075f94dd4949b3275de6174bb301d18012423804b436790d
Status:	Success
Block:	10853763 1 Block Confirmation
Timestamp:	46 secs ago (Aug-17-2021 07:52:58 AM +UTC)
From:	0x05355d84e86e9bab0d18d0479b3a37e70450d4f3
To:	Contract 0xefa28233838f46a80aaac8c309077a9ba70d123a
Tokens Transferred:	From 0x05355d84e86e9... To 0xefa28233838f46... For 1 THORChain ET... (RUNE)
Value:	0.000000000000001 Ether ($0.00)
Transaction Fee:	0.000263506920390962 Ether ($0.00)
Gas Price:	0.000000002844817606 Ether (2.844817606 Gwei)
Txn Type:	0 (Legacy)
Gas Limit:	107,369
Gas Used by Transaction:	92,627 (86.27%)
Base Fee Per Gas:	0.000000001724950431 Ether (1.724950431 Gwei)
Burnt Fees:	0.000159776983572237 Ether
Nonce Position	101 28

Input Data:

#	Name	Type	Data
0	vault	address	0x3d6FB2051192737f27b629503F6247bfB2A9256e
1	asset	address	0x4D704ddA8099305EE9803F02343129BF6BA55FF1
2	amount	uint256	100000000000000000
3	memo	string	SWITCH:tthor1wg5c4rhgk6ayy70ep85k3jlp9xruxqafg508gt

Switch Back

Bifrost Observation



Transaction Hash:	0x2072394ee2ef88cd075f94dd4949b3275de6174bb301d18012423804b436790d
Status:	Success
Block:	10853763 1 Block Confirmation
Timestamp:	46 secs ago (Aug-17-2021 07:52:58 AM +UTC)
From:	0x05355d84e86e9bab0d18d0479b3a37e70450d4f3
To:	Contract 0xefa28233838f46a80aaac8c309077a9ba70d123a
Tokens Transferred:	From 0x05355d84e86e9… To 0xefa28233838f46… For 1 THORChain ET... (RUNE)
Value:	0.000000000000001 Ether ($0.00)
Transaction Fee:	0.000263506920390962 Ether ($0.00)
Gas Price:	0.000000002844817606 Ether (2.844817606 Gwei)
Txn Type:	0 (Legacy)
Gas Limit:	107,369
Gas Used by Transaction:	92,627 (86.27%)
Base Fee Per Gas:	0.000000001724950431 Ether (1.724950431 Gwei)
Burnt Fees:	0.000159776983572237 Ether
Nonce Position	101 28

Input Data:

#	Name	Type	Data
0	vault	address	0x3d6FB2051192737f27b629503F6247bfB2A9256e
1	asset	address	0x4D704ddA8099305EE9803F02343129BF6BA55FF1
2	amount	uint256	100000000000000000
3	memo	string	SWITCH:tthor1wg5c4rhgk6ayy70ep85k3jlp9xruxqafg508gt

Switch Back

Router Deposit

```
>>> router.deposit(VAULT_ADDRESS, ERC20Token[-1], web3.toWei(1, 'ether'), "SWITCH:tthor1wg5c4rhgk6ayy70ep85k3jlp9xruxqafg508gt", {'from':accounts[0], 'value': 10000})
Transaction sent: 0x2072394ee2ef88cd075f94dd4949b3275de6174bb301d18012423804b436790d
  Gas price: 2.844817606 gwei   Gas limit: 107369   Nonce: 101
  THORChain_Router.deposit confirmed - Block: 10853763   Gas used: 92627 (86.27%)

<Transaction '0x2072394ee2ef88cd075f94dd4949b3275de6174bb301d18012423804b436790d'>
>>> ERC20Token[-1].name()
'THORChain ETH.RUNE'
>>>
```

Listing 50

```
1 // IsRune is a helper function ,return true only when the asset
      represent RUNE
2 func (a Asset) IsRune() bool {
3     return a.Equals(BEP2RuneAsset()) || a.Equals(RuneNative) || a.
          Equals(ERC20RuneAsset())
4 }
```

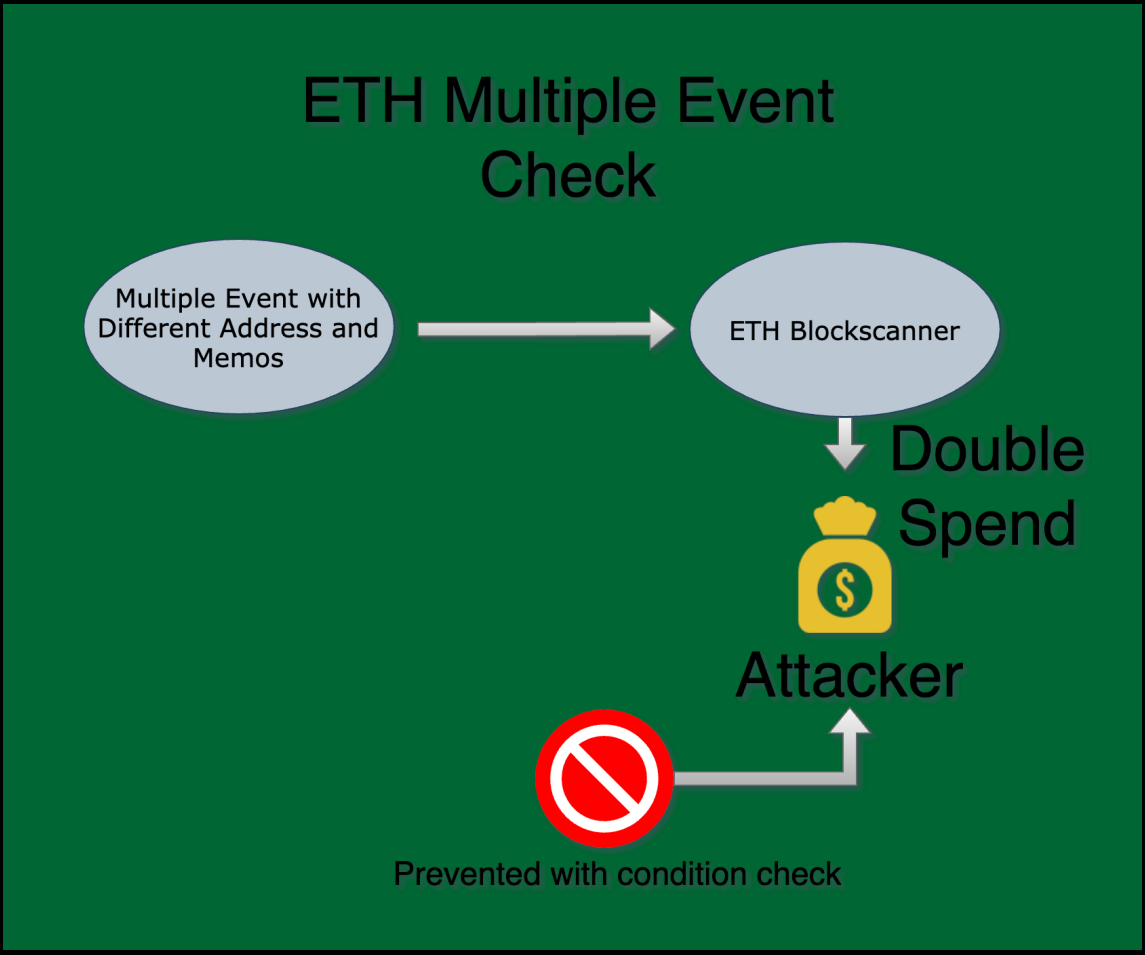
The manipulation tests have been failed due to these protection

mechanisms. The protection mechanism placed on the memos.

MULTIPLE EVENTS CHECK:

Description:

It has been observed that multiple events with different memos are not handled by the code on the previous commits. After the previous hack, the community added multiple protection mechanism for preventing multiple events in the one transaction.



Code Location:

Listing 51: ethereum-block-scanner.go (Lines)

```
840     ...
841 if len(txInItem.Sender) > 0 && !strings.EqualFold(txInItem.Sender,
        transferOutEvt.Vault.String()) {
842         return nil, fmt.Errorf("transfer out event , vault address
                is not the same as sender, ignore")
843 }
844 txInItem.Sender = transferOutEvt.Vault.String()
845 if len(txInItem.To) > 0 && !strings.EqualFold(txInItem.To,
       transferOutEvt.To.String()) {
846     return nil, fmt.Errorf("multiple events in the same
           transaction , have different to addresses , ignore")
847     }
848 txInItem.To = transferOutEvt.To.String()
849 if len(txInItem.Memo) > 0 && !strings.EqualFold(txInItem.Memo,
       transferOutEvt.Memo) {
850     return nil, fmt.Errorf("multiple events in the same
           transaction , have different memo , ignore")
851 }
852     ...
```

The manipulation tests have been failed due to these protection mechanisms. The protection mechanism placed on the events.

4.9 BIFROST STATIC ANALYSIS REPORT

BIFROST STATIC ANALYSIS:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped component. Among the tools used were staticcheck, gosec ineffassign and others. 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:

```
[                                    bifrost/pkg/chainclients/ethereum/tokens_db.go:16] - G101 (CWE-798): Potential hardcoded credentials (Confidence: LOW, Severity: HI
GH)
    15:         // prefixTokenMeta declares prefix to use in leveldb to avoid conflicts
  > 16:         prefixTokenMeta = `eth-tokenmeta-`
    17: )


[                                    bifrost/pkg/chainclients/litecoin/client.go:747] - G601 (CWE-118): Implicit memory aliasing in for loop. (Confidence: MEDIUM, Sever
ity: MEDIUM)
    746:                c.removeFromMemPoolCache(tx.Hash)
  > 747:                txInItem, err := c.getTxIn(&tx, block.Height)
    748:                if err != nil {


[                                    bifrost/pkg/chainclients/dogecoin/client.go:783] - G601 (CWE-118): Implicit memory aliasing in for loop. (Confidence: MEDIUM, Sever
ity: MEDIUM)
    782:                c.removeFromMemPoolCache(tx.Hash)
  > 783:                txInItem, err := c.getTxIn(&tx, block.Height)
    784:                if err != nil {


[                                    bifrost/pkg/chainclients/bitcoincash/client.go:766] - G601 (CWE-118): Implicit memory aliasing in for loop. (Confidence: MEDIUM, Se
verity: MEDIUM)
    765:                c.removeFromMemPoolCache(tx.Hash)
  > 766:                txInItem, err := c.getTxIn(&tx, block.Height)
    767:                if err != nil {


[                                    bifrost/pkg/chainclients/bitcoin/client.go:757] - G601 (CWE-118): Implicit memory aliasing in for loop. (Confidence: MEDIUM, Severi
ty: MEDIUM)
    756:                c.removeFromMemPoolCache(tx.Hash)
  > 757:                txInItem, err := c.getTxIn(&tx, block.Height)
    758:                if err != nil {
```

Staticcheck - Security Analysis Output Sample:

```
blockscanner/blockscanner.go:162:7: should use time.Since instead of time.Now().Sub (S1012)
blockscanner/blockscanner_test.go:170:2: this value of err is never used (SA4006)
blockscanner/types/types.go:5:5: error var UnavailableBlock should have name of the form ErrFoo (ST1012)
blockscanner/types/types.go:6:5: error var FailOutputMatchCriteria should have name of the form ErrFoo (ST1012)
observer/observe.go:17:2: package "gitlab.com/thorchain/thornode/x/thorchain/memo" is being imported more than once (ST1019)
        observer/observe.go:26:2: other import of "gitlab.com/thorchain/thornode/x/thorchain/memo"
observer/observe.go:98:15: error strings should not be capitalized (ST1005)
observer/observe.go:254:20: func (*Observer).isOutboundMsg is unused (U1000)
observer/observe.go:256:2: should use 'return matchOutbound' instead of 'if matchOutbound { return true }; return false' (S1008)
pkg/chainclients/binance/binance.go:20:2: package "gitlab.com/thorchain/binance-sdk/common/types" is being imported more than once (ST1019)
        pkg/chainclients/binance/binance.go:21:2: other import of "gitlab.com/thorchain/binance-sdk/common/types"
pkg/chainclients/binance/binance_block_scanner_test.go:394:2: this value of err is never used (SA4006)
pkg/chainclients/binance/binance_test.go:258:10: empty branch (SA9003)
pkg/chainclients/binance/key_manager.go:32:14: error strings should not be capitalized (ST1005)
pkg/chainclients/binance/metadata.go:23:50: should use make(map[common.PubKey]BinanceMetadata) instead (S1019)
pkg/chainclients/bitcoin/block_meta_accessor_test.go:63:3: should use 'return !meta.TransactionHashExist(hash.String())' instead of 'if meta.TransactionHashExist(hash.String()) { return false }; return true' (S1008)
pkg/chainclients/bitcoin/client.go:254:5: should use time.Since instead of time.Now().Sub (S1012)
pkg/chainclients/bitcoin/signer_test.go:130:2: this value of err is never used (SA4006)
pkg/chainclients/bitcoincash/block_meta_accessor_test.go:63:3: should use 'return !meta.TransactionHashExist(hash.String())' instead of 'if meta.TransactionHashExist(hash.String()) { return false }; return true' (S1008)
pkg/chainclients/bitcoincash/client.go:251:5: should use time.Since instead of time.Now().Sub (S1012)
pkg/chainclients/bitcoincash/signer_test.go:127:2: this value of err is never used (SA4006)
pkg/chainclients/dogecoin/block_meta_accessor_test.go:63:3: should use 'return !meta.TransactionHashExist(hash.String())' instead of 'if meta.TransactionHashExist(hash.String()) { return false }; return true' (S1008)
pkg/chainclients/dogecoin/client.go:251:5: should use time.Since instead of time.Now().Sub (S1012)
pkg/chainclients/dogecoin/signer_test.go:127:2: this value of err is never used (SA4006)
pkg/chainclients/dummy.go:10:5: error var kaboom should have name of the form errFoo (ST1012)
pkg/chainclients/dummy.go:10:14: error strings should not end with punctuation or a newline (ST1005)
pkg/chainclients/ethereum/ethereum.go:777:5: should use time.Since instead of time.Now().Sub (S1012)
pkg/chainclients/ethereum/ethereum_block_scanner_test.go:567:2: this value of err is never used (SA4006)
pkg/chainclients/ethereum/ethereum_test.go:108:4: this value of err is never used (SA4006)
pkg/chainclients/ethereum/ethereum_test.go:638:2: this value of err is never used (SA4006)
pkg/chainclients/ethereum/key_sign_wrapper_test.go:43:2: this value of err is never used (SA4006)
pkg/chainclients/ethereum/key_sign_wrapper_test.go:74:2: this value of buf is never used (SA4006)
pkg/chainclients/ethereum/unstuck_test.go:82:4: this value of err is never used (SA4006)
pkg/chainclients/litecoin/block_meta_accessor_test.go:63:3: should use 'return !meta.TransactionHashExist(hash.String())' instead of 'if meta.TransactionHashExist(hash.String()) { return false }; return true' (S1008)
pkg/chainclients/litecoin/client.go:252:5: should use time.Since instead of time.Now().Sub (S1012)
pkg/chainclients/litecoin/signer_test.go:128:2: this value of err is never used (SA4006)
pkg/chainclients/loadchains.go:28:47: should use make(map[common.Chain]ChainClient) instead (S1019)
pubkeymanager/mock_pool_address_validator.go:38:38: func (*MockPoolAddressValidator).fetchPubKeys is unused (U1000)
pubkeymanager/mock_pool_address_validator.go:49:83: error strings should not end with punctuation or a newline (ST1005)
pubkeymanager/mock_pool_address_validator.go:50:83: error strings should not end with punctuation or a newline (ST1005)
signer/sign_test.go:131:10: empty branch (SA9003)
signer/storage.go:236:45: should use make(map[string][]TxOutStoreItem) instead (S1019)
signer/thorchain_block_scanner.go:31:2: field m is unused (U1000)
signer/thorchain_block_scanner_test.go:113:10: empty branch (SA9003)
thorclient/thorchain.go:176:27: func (*ThorchainBridge).post is unused (U1000)
thorclient/thorchain.go:310:11: error strings should not be capitalized (ST1005)
thorclient/thorchain_test.go:12:2: package "github.com/cosmos/cosmos-sdk/crypto/keyring" is being imported more than once (ST1019)
        thorclient/thorchain_test.go:13:2: other import of "github.com/cosmos/cosmos-sdk/crypto/keyring"
thorclient/types/mode.go:37:20: error strings should not be capitalized (ST1005)
thorclient/types/set_tx.go:13:16: legacytypes.StdSignature is deprecated: StdSignature represents a sig  (SA1019)
tss/keygen_test.go:74:2: this value of err is never used (SA4006)
```

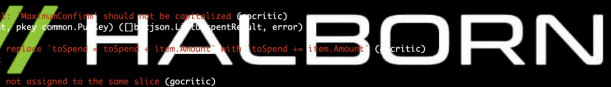

Ineffassign - Security Analysis Output Sample:

```
Pro bifrost % ineffassign ./...
thornode-develop/bifrost/blockscanner/blockscanner_test.go:170:10: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/binance/binance_block_scanner_test.go:394:10: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/bitcoin/signer_test.go:130:12: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/bitcoincash/signer_test.go:127:12: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/dogecoin/signer_test.go:127:12: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/ethereum/ethereum_block_scanner_test.go:567:11: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/ethereum/ethereum_test.go:108:4: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/ethereum/ethereum_test.go:638:19: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/ethereum/key_sign_wrapper_test.go:43:14: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/ethereum/key_sign_wrapper_test.go:74:2: ineffectual assignment to buf
thornode-develop/bifrost/pkg/chainclients/ethereum/unstuck_test.go:82:4: ineffectual assignment to err
thornode-develop/bifrost/pkg/chainclients/litecoin/signer_test.go:128:12: ineffectual assignment to err
thornode-develop/bifrost/tss/keygen_test.go:74:5: ineffectual assignment to err
Pro bifrost %
```

Gocritic - Security Analysis Output Sample:

```
[redacted] bifrost % golangci-lint run ./... -E gocritic
WARN [runner] The linter 'golint' is deprecated (since v1.41.0) due to: The repository of the linter has been archived by the owner.  Replaced by revive.
thorclient/thorchain.go:285:3: ifElseChain: rewrite if-else to switch statement (gocritic)
                if tx.Tx.ToAddress.Equals(obAddr) && !inbound.Contains(tx) {
                ^
blockscanner/blockscanner.go:175:4: commentFormatting: put a space between `//` and comment text (gocritic)
                        //b.logger.Debug().Int64("block height", currentBlock).Msg("fetch txs")
                        ^
tss/mock_thorchain_keymanager.go:60:3: ifElseChain: rewrite if-else to switch statement (gocritic)
                if msgToSign == "BMxXf+K+1dYu3qGgvH59GXoxwwFfTnLjB7hHf3qflPk=" {
                ^
pkg/chainclients/binance/binance.go:173:3: exitAfterDefer: log.Fatal will exit, and `defer func(){...}(...)` will not run (gocritic)
                log.Fatal().Err(err).Msg("fail to read body")
                ^
pkg/chainclients/bitcoin/client.go:205:39: captLocal: `MaximumConfirm' should not be capitalized (gocritic)
func (c *Client) getUTXOs(minConfirm, MaximumConfirm int, pkey common.PubKey) ([]btcjson.ListUnspentResult, error) {
                                      ^
pkg/chainclients/bitcoin/signer.go:126:4: assignOp: replace `toSpend = toSpend + item.Amount` with `toSpend += item.Amount` (gocritic)
                        toSpend = toSpend + item.Amount
                        ^
pkg/chainclients/litecoin/client.go:203:39: captLocal: `MaximumConfirm' should not be capitalized (gocritic)
func (c *Client) getUTXOs(minConfirm, MaximumConfirm int, pkey common.PubKey) ([]btcjson.ListUnspentResult, error) {

pkg/chainclients/litecoin/signer.go:122:4: assignOp: replace `toSpend = toSpend + item.Amount` with `toSpend += item.Amount` (gocritic)
                        toSpend = toSpend + item.Amount
                        ^
pkg/chainclients/bitcoincash/client.go:203:39: captLocal: `MaximumConfirm' should not be capitalized (gocritic)
func (c *Client) getUTXOs(minConfirm, MaximumConfirm int, pkey common.PubKey) ([]btcjson.ListUnspentResult, error) {
                                      ^
pkg/chainclients/bitcoincash/signer.go:126:4: assignOp: replace `toSpend = toSpend + item.Amount` with `toSpend += item.Amount` (gocritic)
                        toSpend = toSpend + item.Amount
                        ^
observer/observe.go:423:33: appendAssign: append result not assigned to the same slice (gocritic)
                        o.onDeck[deckIdx].TxArray = append(txIn.TxArray[:idx], txIn.TxArray[idx+1:]...)
                                                    ^
observer/observe.go:582:4: assignOp: replace `height = height + txIn.ConfirmationRequired` with `height += txIn.ConfirmationRequired` (gocritic)
                        height = height + txIn.ConfirmationRequired
                        ^
[redacted] bifrost %
```



According to the test results, some findings found by these tools were considered as false positives while some of these findings were real security concerns. All relevant findings were reviewed by the auditors and relevant findings addressed in the report as security concerns.

ROUTER SMART CONTRACT

5.1 (HAL-25) SMART CONTRACT - INCOMPLETE EVENT - LOW

Description:

In the THORChain_Router.sol contract, the deposit function emits events after the progress. During the examination of previous attacks, It has been observed that all Bifrost component attacks came from an attacker contracts. Events are useful for inspecting unintended behaviours.

Code Location:

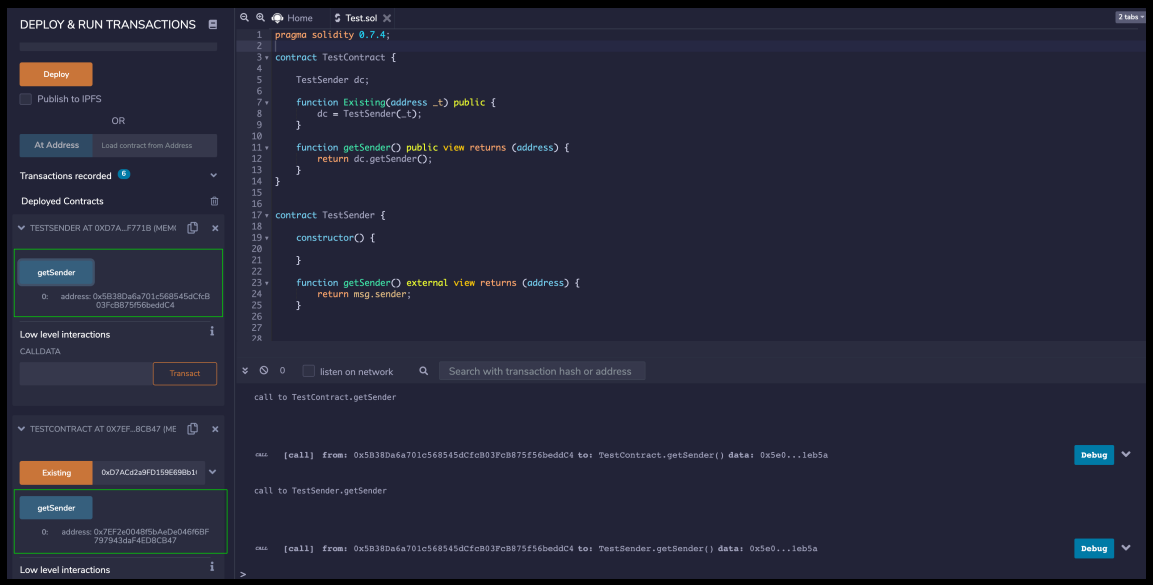
THORChain_Router.sol Line #~68

Listing 52: THORChain-Router.sol (Lines 82)

```
68     function deposit(address payable vault, address asset, uint
           amount, string memory memo) public payable nonReentrant{
69         uint safeAmount;
70         if(asset == address(0)){
71             safeAmount = msg.value;
72             (bool success,) = vault.call{value:safeAmount}("");
73             require(success);
74         } else if(asset == RUNE) {
75             safeAmount = amount;
76             iRUNE(RUNE).transferTo(address(this), amount);
77             iERC20(RUNE).burn(amount);
78         } else {
79             safeAmount = safeTransferFrom(asset, amount); //
                   Transfer asset
80             vaultAllowance[vault][asset] += safeAmount; // Credit
                   to chosen vault
81         }
82         emit Deposit(vault, asset, safeAmount, memo);
83     }
```

However, As the better way for emitting events msg.sender parameter can be added into emit function.

Example Contract



Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider adding msg.sender parameter on the Deposit event. That will provide further information on the debugging process.

Remediation plan:

RISK ACCEPTED: Thorchain Team decided to continue with the current event parameters.

5.2 (HAL-26) SMART CONTRACT - USE OF INSECURE TRANSFERS - LOW

Description:

It is good to add a require() statement that checks the return value of token transfers, or to use something like OpenZeppelin's safeTransfer/safeTransferFrom unless one is sure the given token reverts in case of a failure. Failure to do so will cause silent failures of transfers and affect token accounting in contract.

Code Location:

THORChain_Router.sol Line #~76-77

Listing 53: THORChain-Router.sol (Lines 76)

```
68     function deposit(address payable vault, address asset, uint
           amount, string memory memo) public payable nonReentrant{
69         uint safeAmount;
70         if(asset == address(0)){
71             safeAmount = msg.value;
72             (bool success,) = vault.call{value:safeAmount}("");
73             require(success);
74         } else if(asset == RUNE) {
75             safeAmount = amount;
76             iRUNE(RUNE).transferTo(address(this), amount);
77             iERC20(RUNE).burn(amount);
78         } else {
79             safeAmount = safeTransferFrom(asset, amount); //
                   Transfer asset
80             vaultAllowance[vault][asset] += safeAmount; // Credit
                   to chosen vault
81         }
82         emit Deposit(vault, asset, safeAmount, memo);
83     }
```

Recommendation:

Use SafeERC20 on both networks when possible and ensure that the **transfer/transferFrom** return value is checked. A custom function named safeTransfer can be implemented that checks the return value of the transferTo function and makes sure that the funds were transferred.

Remediation plan:

FUTURE RELEASE: The Thorchain Team will consider implement fix on the future release.

5.3 (HAL-27) SMART CONTRACT - IGNORED RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In the LiquidityFactory contract, there are a few instances where the multiple methods are called and the return value (bool) is ignored.

Code Location:

THORChain_Router.sol Line #~257,258,270

Listing 54: THORChain-Router.sol (Lines 161)

```
157         function _routerDeposit(address _router, address _vault,
                address _asset, uint _amount, string memory _memo)
                internal {
158          vaultAllowance[msg.sender][_asset] -= _amount;
159          (bool success,) = _asset.call(abi.encodeWithSignature("
                approve(address,uint256)", _router, _amount)); //
                Approve to transfer
160          require(success);
161          iROUTER(_router).depositWithExpiry(_vault, _asset, _amount
                , _memo, type(uint).max); // Transfer by depositing
162      }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Add a return value check to avoid an unexpected crash of the contract. Return value checks provide better exception handling.

Remediation plan:

RISK ACCEPTED: Thorchain Team decided to continue without amount return value check.

5.4 (HAL-28) SMART CONTRACT - MISSING AMOUNT CHECK - LOW

Description:

The THORChain_Router contract does not validate amount on the **safeTransferFrom** function. The amount of validation should be added into **safeTransferFrom** function.

Risk Level:

Likelihood - 2
Impact - 2

Code Location:

Listing 55: THORChain-Router.sol

```
143     function safeTransferFrom(address _asset, uint _amount)
            internal returns(uint amount) {
144         uint _startBal = iERC20(_asset).balanceOf(address(this));
145         (bool success,) = _asset.call(abi.encodeWithSignature("
                transferFrom(address,address,uint256)", msg.sender,
                address(this), _amount));
146         require(success);
147         return (iERC20(_asset).balanceOf(address(this)) -
                _startBal);
148     }
```

Recommendation:

Consider to add amount of validation on the **safeTransferFrom** function.

For example:

Listing 56: Example Check (Lines)

```
1     require(amount > 0), "Amount should be more than zero");
```

Remediation plan:

RISK ACCEPTED: Thorchain Team decided to continue without amount of validation.

5.5 (HAL-29) SMART CONTRACT - LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR - INFORMATIONAL

Description:

The THORChain_Router contract has a lack a safety check inside their constructors and functions. Setters of address type parameters should include a zero-address check.

Risk Level:

Likelihood - 1
Impact - 2

Code Location:

Listing 57: THORChain-Router.sol (Lines 57)

```
56     constructor(address rune) {
57         RUNE = rune;
58         _status = _NOT_ENTERED;
59     }
```

Recommendation:

Add proper address validation when assigning a value to a variable from user-supplied data. Better yet, address white-listing/black-listing should be implemented in relevant functions if possible.

For example:

Listing 58: Modifier.sol (Lines 2,3,4)

```
1  modifier validAddress(address addr) {
2      require(addr != address(0), "Address cannot be 0x0");
3      require(addr != address(this), "Address cannot be contract");
4      _;
5  }
```

Remediation plan:

RISK ACCEPTED: Thorchain Team decided to continue without address validation.

5.6 (HAL-30) SMART CONTRACT - PRAGMA TOO RECENT - INFORMATIONAL

Description:

The THORChain_Router contract uses one of the latest pragma version (0.8.0) which was released on December 16, 2020. The latest pragma version (0.8.7) was released in August 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

Reference: https://github.com/ethereum/solidity/releases

In the Solitidy Github repository, there is a json file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Reference

Code Location:

Listing 59: (Lines 5)

```
5 pragma solidity 0.8.3;
6 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

Remediation plan:

FUTURE RELEASE: The Thorchain Team will consider implement fix on the future release.

5.7 (HAL-31) SMART CONTRACT - USAGE OF BLOCK-TIMESTAMP - INFORMATIONAL

Description:

During a manual static review, the tester noticed the use of block.timestamp in THORChain_Router.sol. The contract developers should be aware that this does not mean current time. Now is an alias for block.timestamp. The value of block.timestamp can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles. Miners can influence the timestamp by a tolerance of 900 seconds.

Code Location:

THORChain_Router.sol

Listing 60: THORChain-Router.sol (Lines 178)

```
174     function depositWithExpiry(address payable vault, address
            asset, uint amount, string memory memo, uint expiration)
            external payable {
175         require(block.timestamp < expiration, "THORChain_Router:
                expired");
176         deposit(vault, asset, amount, memo);
177     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years,

days and months rather than seconds.

Remediation plan:

ACKNOWLEDGED: Thorchain Team is aware of the issue. The fix will be considered on the future implementations.

5.8 (HAL-32) SMART CONTRACT - FOR LOOP OVER DYNAMIC ARRAY - INFORMATIONAL

Description:

Calls inside a loop might lead to a denial-of-service attack. The function discovered is a for loop on the variable that iterates up to the coins length. If this integer is evaluated at extremely large numbers,this can cause a DoS.

Code Location:

THORChain_Router.sol Line #143

Listing 61: THORChain-Router.sol (Lines)

```
143     function returnVaultAssets(address router, address payable
            asgard, Coin[] memory coins, string memory memo) external
            payable {
144         if (router == address(this)){
145             for(uint i = 0; i < coins.length; i++){
146                 _adjustAllowances(asgard, coins[i].asset, coins[i
                        ].amount);
147             }
148             emit VaultTransfer(msg.sender, asgard, coins, memo);
                    // Does not include ETH.
149         } else {
150             for(uint i = 0; i < coins.length; i++){
151                 _routerDeposit(router, asgard, coins[i].asset,
                        coins[i].amount, memo);
152             }
153         }
154         (bool success,) = asgard.call{value:msg.value}(""); //ETH
                amount needs to be parsed from tx.
155         require(success);
156     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If possible, use pull over push strategy for external calls.

Remediation plan:

ACKNOWLEDGED: Thorchain Team is aware of the issue. The fix will be considered on the future implementations.

5.9 (HAL-33) SMART CONTRACT - NO TEST COVERAGE - INFORMATIONAL

Description:

Unlike traditional software, smart contracts cannot be modified unless deployed using a proxy contract. Because of the permanence, unit tests and functional testing are recommended to ensure the code works correctly before deployment. Mocha and Chai are valuable tools to perform unit tests in smart contracts. Mocha is a JavaScript testing framework for creating synchronous and asynchronous unit tests, and Chai is a library with assertion functionality such as assert or expect and should be used to develop custom unit tests.

References:
https://github.com/mochajs/mocha
https://github.com/chaijs/chai
https://docs.openzeppelin.com/learn/writing-automated-tests

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended performing as many test cases as possible to cover all conceivable scenarios in the smart contract.

Remediation plan:

SOLVED: Thorchain Team is implemented test cases on the contract.

5.10 SMART CONTRACT AUTOMATED TESTING

STATIC ANALYSIS REPORT:

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
INFO:Detectors:
THORChain_Router.deposit(address,address,uint256,string) (THORChain_Router.sol#68-83) ignores return value by iRUNE(RUNE).transferTo(address(this),amount) (THORChain_Router.sol#76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
THORChain_Router.constructor(address).rune (THORChain_Router.sol#56) lacks a zero-check on :
                - RUNE = rune (THORChain_Router.sol#57)
THORChain_Router.deposit(address,address,uint256,string).vault (THORChain_Router.sol#68) lacks a zero-check on :
                - (success) = vault.call{value: safeAmount}() (THORChain_Router.sol#72)
THORChain_Router.transferOut(address,address,uint256,string).to (THORChain_Router.sol#100) lacks a zero-check on :
                - (success,None) = to.call{value: msg.value}() (THORChain_Router.sol#104)
                - (success,None) = asset.call(abi.encodeWithSignature(transfer(address,uint256),to,amount)) (THORChain_Router.sol#107)
THORChain_Router.returnVaultAssets(address,address,THORChain_Router.Coin[],string).asgard (THORChain_Router.sol#125) lacks a zero-check on :
                - (success) = asgard.call{value: msg.value}() (THORChain_Router.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in THORChain_Router.deposit(address,address,uint256,string) (THORChain_Router.sol#68-83):
        External calls:
        - safeAmount = safeTransferFrom(asset,amount) (THORChain_Router.sol#79)
                - (success) = _asset.call(abi.encodeWithSignature(transferFrom(address,address,uint256),msg.sender,address(this),_amount)) (THORChain_Router.sol#145)
        State variables written after the call(s):
        - vaultAllowance[vault][asset] += safeAmount (THORChain_Router.sol#80)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in THORChain_Router.deposit(address,address,uint256,string) (THORChain_Router.sol#68-83):
        External calls:
        - (success) = vault.call{value: safeAmount}() (THORChain_Router.sol#72)
        - iRUNE(RUNE).transferTo(address(this),amount) (THORChain_Router.sol#76)
        - iERC20(RUNE).burn(amount) (THORChain_Router.sol#77)
        - safeAmount = safeTransferFrom(asset,amount) (THORChain_Router.sol#79)
                - (success) = _asset.call(abi.encodeWithSignature(transferFrom(address,address,uint256),msg.sender,address(this),_amount)) (THORChain_Router.sol#145)
        External calls sending eth:
        - (success) = vault.call{value: safeAmount}() (THORChain_Router.sol#72)
        Event emitted after the call(s):
        - Deposit(vault,asset,safeAmount,memo) (THORChain_Router.sol#82)
Reentrancy in THORChain_Router.transferOut(address,address,uint256,string) (THORChain_Router.sol#100-112):
        External calls:
        - (success,None) = to.call{value: msg.value}() (THORChain_Router.sol#104)
        - (success,None) = asset.call(abi.encodeWithSignature(transfer(address,uint256),to,amount)) (THORChain_Router.sol#107)
        External calls sending eth:
        - (success,None) = to.call{value: msg.value}() (THORChain_Router.sol#104)
        Event emitted after the call(s):
        - TransferOut(msg.sender,to,asset,safeAmount,memo) (THORChain_Router.sol#111)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
THORChain_Router.depositWithExpiry(address,address,uint256,string,uint256) (THORChain_Router.sol#62-65) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < expiration,THORChain_Router: expired) (THORChain_Router.sol#63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Pragma version0.8.3 (THORChain_Router.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.3 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in THORChain_Router.deposit(address,address,uint256,string) (THORChain_Router.sol#68-83):
        - (success) = vault.call{value: safeAmount}() (THORChain_Router.sol#72)
Low level call in THORChain_Router.transferOut(address,address,uint256,string) (THORChain_Router.sol#100-112):
        - (success,None) = to.call{value: msg.value}() (THORChain_Router.sol#104)
        - (success,None) = asset.call(abi.encodeWithSignature(transfer(address,uint256),to,amount)) (THORChain_Router.sol#107)
Low level call in THORChain_Router.returnVaultAssets(address,address,THORChain_Router.Coin[],string) (THORChain_Router.sol#125-138):
        - (success) = asgard.call{value: msg.value}() (THORChain_Router.sol#136)
Low level call in THORChain_Router.safeTransferFrom(address,uint256) (THORChain_Router.sol#143-148):
        - (success) = _asset.call(abi.encodeWithSignature(transferFrom(address,address,uint256),msg.sender,address(this),_amount)) (THORChain_Router.sol#145)
Low level call in THORChain_Router._routerDeposit(address,address,address,uint256,string) (THORChain_Router.sol#157-162):
        - (success) = _asset.call(abi.encodeWithSignature(approve(address,uint256),_router,_amount)) (THORChain_Router.sol#159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
THORChain_Router (THORChain_Router.sol#22-163) should inherit from iROUTER (THORChain_Router.sol#17-19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
Contract iERC20 (THORChain_Router.sol#8-11) is not in CapWords
Contract iRUNE (THORChain_Router.sol#13-15) is not in CapWords
Contract iROUTER (THORChain_Router.sol#17-19) is not in CapWords
Contract THORChain_Router (THORChain_Router.sol#22-163) is not in CapWords
Parameter THORChain_Router.safeTransferFrom(address,uint256)._asset (THORChain_Router.sol#143) is not in mixedCase
Parameter THORChain_Router.safeTransferFrom(address,uint256)._amount (THORChain_Router.sol#143) is not in mixedCase
Variable THORChain_Router.RUNE (THORChain_Router.sol#23) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Slither:THORChain_Router.sol analyzed (4 contracts with 72 detectors), 24 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

5.11 SMART CONTRACT AUTOMATED SECURITY SCAN RESULTS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and identify low-hanging fruit on the scoped contract targeted for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine, and sent the compiled results to MythX to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with the THORChain-Router.sol.

Results:

Report for contracts/THORChain_Router.sol
https://dashboard.mythx.io/#/console/analyses/02f1fd5a-4e01-4fce-b051-6930042d1e5e

Line	SWC Title	Severity	Short Description
22	(SWC-123) Requirement Violation	Low	Requirement violation.
53	(SWC-107) Reentrancy	Low	Write to persistent state following external call.
144	(SWC-123) Requirement Violation	Low	Requirement violation.
161	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.

THANK YOU FOR CHOOSING

// HALBORN