Thorchain TSS

Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 1st, 2021 - September 30th, 2021**

Visit: **Halborn.com**

DRAFT

DRAFT

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/15/2021	Gokberk Gulgun
0.5	Document Updates	09/25/2021	Gokberk Gulgun
1.0	Final Review	09/30/2021	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

ThorChain engaged Halborn to conduct a security assessment on their **TSS** repository beginning on September 1st, 2021, and ending September 30th, 2021. Halborn was provided access to the source code of the application and the testing environment in order 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.

In summary, Halborn identified several security risks that need to be addressed.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding 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:

TSS:

- 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 TSS functionalities.
- Dynamic Analysis.

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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

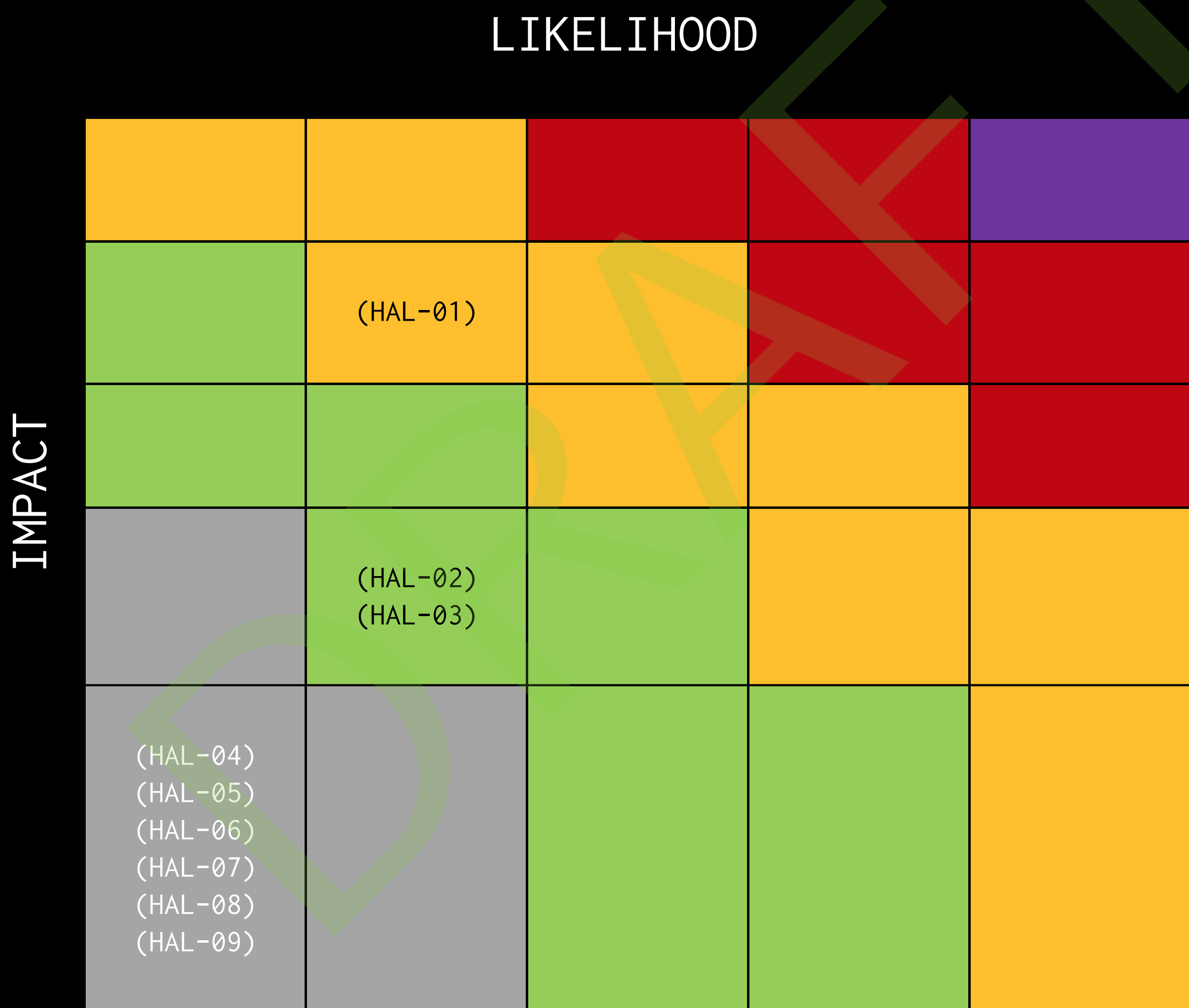
1.4 SCOPE

IN-SCOPE : The repository concerned is: **https://gitlab.com/thorchain/tss/go-tss**

Commit ID : 29f841a0d1d38b264eaf71fbdced8afe005c3544.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	6



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INSECURE FILE PERMISSIONS	Medium	-
SENSITIVE INFORMATION IN THE ENVIRONMENT VARIABLES	Low	-
LACK OF MEMORY PROTECTION MECHANISM	Low	-
EXAMINATION OF KEY DERIVATION	Informational	-
USE WEAK RANDOM NUMBER GENERATOR	Informational	-
LACK OF KEY SIZE CHECK	Informational	-
TYPO ON THE ENCRYPTION COMMENT	Informational	-
MISSING GO COMPILER BUILD DIRECTIVES	Informational	-
DOCKER CONTAINER WITH ROOT PRIVILEGES	Informational	-
STATIC ANALYSIS REPORT	-	-

FINDINGS & TECH DETAILS

DRAFT

3.1 (HAL-01) INSECURE FILE PERMISSIONS - MEDIUM

Description:

When a resource is given a permission setting that provides access to a wider range of actors than required, it could lead to the exposure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution, or sensitive user data. On the **TSS**, **os.ModePerm** permission is applied on the file system.

Code Location:

Listing 1: tss-recovery.go

```go

    if len(*export) > 0 && len(*password) >= 8 {
        keyfile, err := exportKeyStore(privKey.Serialize(), *
            password)
        if err != nil {
            fmt.Printf("--->%v", err)
        }

        jsonString, _ := json.Marshal(keyfile)
        err = ioutil.WriteFile(*export, jsonString, os.ModePerm)
        if err != nil {
            fmt.Printf("--->%v", err)
        }
        fmt.Printf("---wrote to: %+v\n", *export)
    }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Set the permissions of the files and directories properly so that unauthorized users cannot access critical resources unnecessarily.

3.2 (HAL-02) SENSITIVE INFORMATION IN THE ENVIRONMENT VARIABLES - LOW

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---suddenly 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.

Docker Compose Go Tss

Listing 2

```
version: '3'

services:
  tss0:
    hostname: tss0
    ports:
      - 8080:8080
      - 6668:6668
    build:
      context: ../
      dockerfile: Dockerfile
    #image: registry.gitlab.com/thorchain/tss/go-tss
    restart: unless-stopped
    environment:
      - PRIVKEY=${TSS_0}
      - NET=testnet
    command: /go/bin/start.bash
    networks:
      localnet:
```

```
        ipv4_address: 192.168.10.1

  tss1:
    hostname: tss1
    ports:
      - 8081:8080
      - 6669:6668
    build:
      context: ../
      dockerfile: Dockerfile
    #image: registry.gitlab.com/thorchain/tss/go-tss
    restart: unless-stopped
    environment:
      - PRIVKEY=${TSS_1}
      - NET=testnet
    depends_on:
      - tss0
    command: /go/bin/start-tss.bash
    networks:
      localnet:
        ipv4_address: 192.168.10.2

  tss2:
    hostname: tss2
    ports:
      - 8082:8080
      - 6667:6668
    build:
      context: ../
      dockerfile: Dockerfile
    #image: registry.gitlab.com/thorchain/tss/go-tss
    restart: unless-stopped
    environment:
      - PRIVKEY=${TSS_2}
      - NET=testnet
    depends_on:
      - tss0
    command: /go/bin/start-tss.bash
    networks:
      localnet:
        ipv4_address: 192.168.10.3
  tss3:
    hostname: tss3
    ports:
```

```
      - 8083:8080
      - 6666:6668
    build:
      context: ../
      dockerfile: Dockerfile
    #image: registry.gitlab.com/thorchain/tss/go-tss
    restart: unless-stopped
    environment:
      - PRIVKEY=${TSS_3}
      - NET=testnet
    depends_on:
      - tss0
    command: /go/bin/start-tss.bash
    networks:
      localnet:
        ipv4_address: 192.168.10.4

networks:
  localnet:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.10.0/16
```


Remediation:

It is recommended to store the **PRIVKEY** in the secure docker storage.

Reference:

https://docs.docker.com/engine/swarm/secrets/

3.3 (HAL-03) LACK OF MEMORY PROTECTION MECHANISM - LOW

Description:

During the manual code review, It has been observed that there is no mechanism implemented for handling sensitive values in memory. Memguard package is designed to allow you to easily handle sensitive values in memory.

Code Location:

cmd/tss/main.go

Listing 3: main.go

```
inBuf := bufio.NewReader(os.Stdin)
priKeyBytes, err := input.GetPassword("input node secret key:", inBuf)
if err != nil {
    fmt.Printf("error in get the secret key: %s\n", err.Error())
    return
}
priKey, err := conversion.GetPriKey(priKeyBytes)
if err != nil {
    log.Fatal(err)
}
```

Reference Memguard

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider using memory protection mechanism on the **TSS** component. Sample **Stdin** protection via memory protection can be seen below.

Listing 4: main.go

```go
package stdin

import (
    "errors"
    "os"

    "github.com/awnumar/memguard"
)

// ReadKeyFromStdin reads a key from standard inputs and returns it sealed inside an Enclave object.
func ReadKeyFromStdin() (*memguard.Enclave, error) {
    key, err := memguard.NewBufferFromReaderUntil(os.Stdin, '\n')
    if err != nil {
        // error encountered before '\n' was reached
        return nil, err
    }
    if key.Size() == 0 {
        return nil, errors.New("no input received")
    }
    return key.Seal(), nil
}
```

3.4 (HAL-04) EXAMINATION OF KEY DERIVATION - INFORMATIONAL

Description:

A key derivation function is useful when encrypting data based on a password or any other not-fully-random data. It uses a pseudorandom function to derive a secure encryption key based on the password. In the TSS, key.go file is implemented for the key derivation and **PKDBF2** has been used. Key derives a key from the password, salt and iteration count, returning a []byte of length keylen that can be used as cryptographic key. The key is derived based on the method described as PBKDF2 with the HMAC variant using the supplied hash function. On the implementation, **SHA256** has been used.

Code Location:

cmd/tss-recovery/key.go

Listing 5: key.go

```
cipherParamsJSON := cipherParams{IV: hex.EncodeToString(iv)}
derivedKey := pbkdf2.Key([]byte(password), salt, 262144, 32,
    sha256.New)
if err != nil {
    return nil, err
}
encryptKey := derivedKey[:32]
cipherText, err := aesCTRXOR(encryptKey, privKey, iv)
if err != nil {
```

Comparison Bcrypt & PKDBF2 & Scrypt:

Reference

14 STRONGER KEY DERIVATION VIA SEQUENTIAL MEMORY-HARD FUNCTIONS

TABLE 1. Estimated cost of hardware to crack a password in 1 year.

KDF	6 letters	8 letters	8 chars	10 chars	40-char text	80-char text
DES CRYPT	< \$1	< \$1	< \$1	< \$1	< \$1	< \$1
MD5	< \$1	< \$1	< \$1	\$1.1k	\$1	\$1.5T
MD5 CRYPT	< \$1	< \$1	\$130	\$1.1M	\$1.4k	$\$1.5 \times 10^{15}$
PBKDF2 (100 ms)	< \$1	< \$1	\$18k	\$160M	\$200k	$\$2.2 \times 10^{17}$
bcrypt (95 ms)	< \$1	\$4	\$130k	\$1.2B	\$1.5M	\$48B
scrypt (64 ms)	< \$1	\$150	\$4.8M	\$43B	\$52M	$\$6 \times 10^{19}$
PBKDF2 (5.0 s)	< \$1	\$29	\$920k	\$8.3B	\$10M	$\$11 \times 10^{18}$
bcrypt (3.0 s)	< \$1	\$130	\$4.3M	\$39B	\$47M	\$1.5T
scrypt (3.8 s)	\$900	\$610k	\$19B	\$175T	\$210B	$\$2.3 \times 10^{23}$

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

On the recent commits, ThorChain Team is decided to continue with scrypt. Scrypt has the added benefit of having extra **RAM/Memory** requirements, making it more GPU-resistant than SHA, BCrypt or PBKDF2.

Remediation:

Fixed on the recent commits.

3.5 (HAL-05) USE WEAK RANDOM NUMBER GENERATOR - INFORMATIONAL

Description:

The random numbers generated could be predicted. The use of a predictable random value can lead to vulnerabilities when used in certain security critical contexts. Using a cryptographically weak pseudo-random number generator may allow an attacker to predict what security-sensitive value will be generated. In the **“tss-helper”** go file, **math/rand** function is used instead of **crypto/rand**. Although, the function is used for the test case, for the critical components like TSS, secure random generator should be used on the repository.

Code Location:

Listing 6: tss-helper.go

```go
import (
	"errors"
	"math/rand"

	"github.com/blang/semver"
	sdk "github.com/cosmos/cosmos-sdk/types"
	atypes "github.com/cosmos/cosmos-sdk/x/auth/vesting/types"
	"github.com/libp2p/go-libp2p-core/peer"
	"github.com/tendermint/tendermint/crypto/secp256k1"
)

// GetRandomPubKey for test
func GetRandomPubKey() string {
	_, pubKey, _ := atypes.KeyTestPubAddr()
	bech32PubKey, _ := sdk.Bech32ifyPubKey(sdk.
		Bech32PubKeyTypeAccPub, pubKey)
	return bech32PubKey
}

// GetRandomPeerID for test
func GetRandomPeerID() peer.ID {
	_, pubKey, _ := atypes.KeyTestPubAddr()
```

```go
    var pk secp256k1.PubKey
    copy(pk[:], pubKey.Bytes())
    peerID, _ := GetPeerIDFromSecp256PubKey(pk)
    return peerID
}

const letterBytes = ”abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ”
const (
    letterIdxBits = 6                    // 6 bits to represent a letter index
    letterIdxMask = 1<<letterIdxBits - 1 // All 1-bits, as many as letterIdxBits
)

func RandStringBytesMask(n int) string {
    b := make([]byte, n)
    for i := 0; i < n; {
        if idx := int(rand.Int63() & letterIdxMask); idx < len(letterBytes) {
            b[i] = letterBytes[idx]
            i++
        }
    }
    return string(b)
}

func VersionLTCheck(currentVer, expectedVer string) (bool, error) {
    c, err := semver.Make(expectedVer)
    if err != nil {
        return false, errors.New(”fail to parse the expected version”)
    }
    v, err := semver.Make(currentVer)
    if err != nil {
        return false, errors.New(”fail to parse the current version”)
    }
    return v.LT(c), nil
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to use crypto/rand instead of math/rand.

3.6 (HAL-06) LACK OF KEY SIZE CHECK - INFORMATIONAL

Description:

On the localstate_mgr, the **AES-GCM** mode has been used for the encryption process. However, on the /common/encryption.go, key size is not validated before a cryptographic operations. The data should be validated according to size.

Code Location:

Listing 7: common/encryption.go

```
func AESEncrypt(data, derivedKey []byte) ([]byte, error) {
    if len(data) == 0 || len(derivedKey) == 0 {
        return nil, errors.New("invalid data or derivedkey")
    }

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

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

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

// AESDecrypt the input data with passphrase
func AESDecrypt(data, derivedKey []byte) ([]byte, error) {
    if len(data) == 0 || len(derivedKey) == 0 {
        return nil, errors.New("invalid data or derivedkey")
    }

```

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

Risk Level:

Likelihood - 1
Impact - 1

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.

3.7 (HAL-07) TYPO ON THE ENCRYPTION COMMENT - INFORMATIONAL

Description:

On the TSS component, Key is encrypted via **AES** Counter method. The Counter (CTR) mode is a typical block cipher mode of operation using block cipher algorithm. The progress have been completed on the key.go file. On the encryption implementation, The following comment is marked as using **AES-128**. However, on the key generation the variable is defined as 32 bytes(256 bit - encryptKey := derivedKey[:32]).

Code Location:

Listing 8: aesCTRXOR.go

```
func aesCTRXOR(key, inText, iv []byte) ([]byte, error) {
    // AES-128 is selected due to size of encryptKey.
    aesBlock, err := aes.NewCipher(key)
    if err != nil {
        return nil, err
    }
    stream := cipher.NewCTR(aesBlock, iv)
    outText := make([]byte, len(inText))
    stream.XORKeyStream(outText, inText)
    return outText, err
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to fix the typo on the code comments.

3.8 (HAL-08) MISSING GO COMPILER BUILD DIRECTIVES - INFORMATIONAL

Description:

During the tests, it has been observed that Go compiler build flags are not configured. The use of compiler flags and compiler sequences can optimize and improve the performance of specific types of applications.

Risk Level:

Likelihood - 1
Impact - 1

Repository Makefile Flags:

Listing 9: DockerFile (Lines)

```
FROM golang:1.15.6-alpine AS builder

RUN apk update && apk add --no-cache git
WORKDIR /go/src/app
COPY . .
RUN GO111MODULE=on go mod download
WORKDIR /go/src/app/cmd/tss
RUN GOOS=linux GOARCH=amd64 go build -ldflags="-w -s" -o tss

#
# Main
#
FROM alpine:latest
ARG privkey
ARG net
ENV PRIVKEY=$privkey
ENV NET=$net
RUN apk add --update ca-certificates curl
RUN mkdir -p /go/bin
COPY --from=builder /go/src/app/cmd/tss /go/bin
COPY build/start-tss.bash /go/bin/start-tss.bash
```

```
COPY build/start.bash /go/bin/start.bash
EXPOSE 6668
EXPOSE 8080
RUN chmod +x /go/bin/start-tss.bash
RUN chmod +x /go/bin/start.bash
```

Example Flags:

Listing 10: Example Compiler Build Flags (Lines)

```
-a
    force rebuilding of packages that are already up to date.
-ldflags "-s -w"
    The -w turns off DWARF debugging information
    The -s turns off generation of the Go symbol table
-trimpath
    The -trimpath Remove all file system paths from the resulting
        executable.
-gcflags
    arguments to pass on each go tool compile invocation.
```

Recommendation:

Enabling compiler build flags could make binary build faster and outputs a smaller and probably more efficient binary. Therefore, the flags should be reviewed and enabled according to the structure.

3.9 (HAL-09) DOCKER CONTAINER WITH ROOT PRIVILEGES - INFORMATIONAL

Description:

Docker containers typically run with root privileges by default. This allows for unrestricted container management, which means you can do things like install system packages, edit config files, bind privileged ports. During the static analysis, it has been observed that docker image is maintained via root user.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

Docker File

Listing 11: DockerFile (Lines)

```
FROM golang:1.15.6-alpine AS builder

RUN apk update && apk add --no-cache git
WORKDIR /go/src/app
COPY . .
RUN GO111MODULE=on go mod download
WORKDIR /go/src/app/cmd/tss
RUN GOOS=linux GOARCH=amd64 go build -ldflags="-w -s" -o tss

#
# Main
#
FROM alpine:latest
ARG privkey
ARG net
ENV PRIVKEY=$privkey
ENV NET=$net
```

```
RUN apk add --update ca-certificates curl
RUN mkdir -p /go/bin
COPY --from=builder /go/src/app/cmd/tss /go/bin
COPY build/start-tss.bash /go/bin/start-tss.bash
COPY build/start.bash /go/bin/start.bash
EXPOSE 6668
EXPOSE 8080
RUN chmod +x /go/bin/start-tss.bash
RUN chmod +x /go/bin/start.bash
```

Recommendation:

It is recommended that to build dockerfile and run container as non-root user.

Listing 12: Reference

```
USER 1001: this is a non-root user UID, and here it is assigned to
    the image to run the current container as an unprivileged user
   . By doing so, the added security and other restrictions
   mentioned above are applied to the container.
```

STATIC ANALYSIS REPORT

4.1 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 13: Rule Set (Lines)

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

```
severity:error rule:dgryski.semgrep-go.oserrors.os-error-handling-functions: New code should use errors.Is with the appropriate error type
70:     if _, err := os.Stat(dir); !os.IsNotExist(err) {

go-tss-master/cmd/tss-benchsign/main.go
severity:error rule:dgryski.semgrep-go.oserrors.os-error-handling-functions: New code should use errors.Is with the appropriate error type
65:     if stat, err := os.Stat(dir); os.IsNotExist(err) || stat == nil || !stat.IsDir() {
--------------------------------------------------------------------------------
69:     if _, err := os.Stat(makeKeyGenDataFilePath(dir, *endQuorum-1)); os.IsNotExist(err) {

go-tss-master/keygen/tss_keygen.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
153:    for {
154:            select {
155:            case <-errChan: // when keyGenParty return
156:                    tKeyGen.logger.Error().Msg("key gen failed")
157:                    return nil, errors.New("error channel closed fail to start local party")
158:
159:            case <-tKeyGen.stopChan: // when TSS processor receive signal to quit
160:                    return nil, errors.New("received exit signal")
161:
162:            case <-time.After(tssConf.KeyGenTimeout):
163:                    // we bail out after KeyGenTimeoutSeconds
164:                    tKeyGen.logger.Error().Msgf("fail to generate message with %s", tssConf.KeyGenTimeout.String())
165:                    lastMsg := blameMgr.GetLastMsg()
166:                    failReason := blameMgr.GetBlame().FailReason
167:                    if failReason == "" {
168:                            failReason = blame.TssTimeout
169:                    }
170:                    if lastMsg == nil {
171:                            tKeyGen.logger.Error().Msg("fail to start the keygen, the last produced message of this node is none")
172:                            return nil, errors.New("timeout before shared message is generated")
173:                    }
174:                    blameNodesUnicast, err := blameMgr.GetUnicastBlame(messages.KEYGEN2aUnicast)
175:                    if err != nil {
176:                            tKeyGen.logger.Error().Err(err).Msg("error in get unicast blame")
177:                    }
178:                    tKeyGen.tssCommonStruct.P2PPeersLock.RLock()
179:                    threshold, err := conversion.GetThreshold(len(tKeyGen.tssCommonStruct.P2PPeers) + 1)
180:                    tKeyGen.tssCommonStruct.P2PPeersLock.RUnlock()
181:                    if err != nil {
182:                            tKeyGen.logger.Error().Err(err).Msg("error in get the threshold to generate blame")
183:                    }
184:
185:                    if len(blameNodesUnicast) > 0 && len(blameNodesUnicast) <= threshold {
186:                            blameMgr.GetBlame().SetBlame(failReason, blameNodesUnicast, true)
187:                    }
188:                    blameNodesBroadcast, err := blameMgr.GetBroadcastBlame(lastMsg.Type())
189:                    if err != nil {
190:                            tKeyGen.logger.Error().Err(err).Msg("error in get broadcast blame")
191:                    }
192:                    blameMgr.GetBlame().AddBlameNodes(blameNodesBroadcast...)
193:
194:                    // if we cannot find the blame node, we check whether everyone send me the share
195:                    if len(blameMgr.GetBlame().BlameNodes) == 0 {
196:                            blameNodesMisingShare, isUnicast, err := blameMgr.TssMissingShareBlame(messages.TSSKEYGENROUNDS)
```

```
go-tss-master/keysign/tss_keysign.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
188:    for {
189:            select {
190:            case <-errChan: // when key sign return
191:                    tKeySign.logger.Error().Msg("key sign failed")
192:                    return nil, errors.New("error channel closed fail to start local party")
193:            case <-tKeySign.stopChan: // when TSS processor receive signal to quit
194:                    return nil, errors.New("received exit signal")
195:            case <-time.After(tssConf.KeySignTimeout):
196:                    // we bail out after KeySignTimeoutSeconds
197:                    tKeySign.logger.Error().Msgf("fail to sign message with %s", tssConf.KeySignTimeout.String())
198:                    lastMsg := blameMgr.GetLastMsg()
199:                    failReason := blameMgr.GetBlame().FailReason
200:                    if failReason == "" {
201:                            failReason = blame.TssTimeout
202:                    }
203:
204:                    tKeySign.tssCommonStruct.P2PPeersLock.RLock()
205:                    threshold, err := conversion.GetThreshold(len(tKeySign.tssCommonStruct.P2PPeers) + 1)
206:                    tKeySign.tssCommonStruct.P2PPeersLock.RUnlock()
207:                    if err != nil {
208:                            tKeySign.logger.Error().Err(err).Msg("error in get the threshold for generate blame")
209:                    }
210:                    if !lastMsg.IsBroadcast() {
211:                            blameNodesUnicast, err := blameMgr.GetUnicastBlame(lastMsg.Type())
212:                            if err != nil {
213:                                    tKeySign.logger.Error().Err(err).Msg("error in get unicast blame")
214:                            }
215:                            if len(blameNodesUnicast) > 0 && len(blameNodesUnicast) <= threshold {
216:                                    blameMgr.GetBlame().SetBlame(failReason, blameNodesUnicast, true)
217:                            }
218:                    } else {
219:                            blameNodesUnicast, err := blameMgr.GetUnicastBlame(conversion.GetPreviousKeySignUicast(lastMsg.Type()))
220:                            if err != nil {
221:                                    tKeySign.logger.Error().Err(err).Msg("error in get unicast blame")
222:                            }
223:                            if len(blameNodesUnicast) > 0 && len(blameNodesUnicast) <= threshold {
224:                                    blameMgr.GetBlame().SetBlame(failReason, blameNodesUnicast, true)
225:                            }
226:                    }
227:
228:                    blameNodesBroadcast, err := blameMgr.GetBroadcastBlame(lastMsg.Type())
229:                    if err != nil {
230:                            tKeySign.logger.Error().Err(err).Msg("error in get broadcast blame")
231:                    }
232:                    blameMgr.GetBlame().AddBlameNodes(blameNodesBroadcast...)
233:
234:                    // if we cannot find the blame node, we check whether everyone send me the share
235:                    if len(blameMgr.GetBlame().BlameNodes) == 0 {
236:                            blameNodesMisingShare, isUnicast, err := blameMgr.TssMissingShareBlame(messages.TSSKEYSIGNROUNDS)
237:                            if err != nil {
238:                                    tKeySign.logger.Error().Err(err).Msg("fail to get the node of missing share ")
239:                            }
240:
241:                            if len(blameNodesMisingShare) > 0 && len(blameNodesMisingShare) <= threshold {

go-tss-master/p2p/party_coordinator.go
severity:error rule:dgryski.semgrep-go.timeafter.leaky-time-after: Leaky use of time.After in for-select
435:            for {
436:                    select {
437:                    case <-peerGroup.notify:
438:                            pc.logger.Debug().Msg("we have enough participants")
439:                            return
440:
441:                    case <-time.After(pc.timeout):
442:                            // timeout
443:                            pc.logger.Error().Msg("leader waits for peers timeout")
444:                            return
445:                    case result := <-sigChan:
446:                            sigNotify = result
447:                    }
448:            }
--------------------------------------------------------------------------------
531:            for {
532:                    select {
533:                    case <-peerGroup.notify:
534:                            pc.logger.Debug().Msg("we have found the new peer")
535:                            if peerGroup.getCoordinationStatus() {
536:                                    close(done)
537:                                    return
538:                            }
539:                    case <-time.After(pc.timeout):
540:                            // timeout
541:                            close(done)
542:                            return
543:                    }
544:            }

go-tss-master/storage/localstate_mgr.go
severity:error rule:dgryski.semgrep-go.oserrors.os-error-handling-functions: New code should use errors.Is with the appropriate error type
50:             if err != nil && os.IsNotExist(err) {
--------------------------------------------------------------------------------
106:    if _, err := os.Stat(filePathName); os.IsNotExist(err) {
go-tss-master/cmd/tss-benchgen/main.go
severity:error rule:trailofbits.go.unchecked-type-assertion.unchecked-type-assertion: Unchecked type assertion.

139:            P := keygen.NewLocalParty(params, outCh, endCh, preParamTestData).(*keygen.LocalParty)

go-tss-master/cmd/tss-benchsign/main.go
severity:error rule:trailofbits.go.unchecked-type-assertion.unchecked-type-assertion: Unchecked type assertion.

135:            P := signing.NewLocalParty(msg, params, keys[i], outCh, endCh).(*signing.LocalParty)

go-tss-master/common/tss.go
severity:error rule:trailofbits.go.unchecked-type-assertion.unchecked-type-assertion: Unchecked type assertion.

269:            localMsgParty := data.(btss.Party)
--------------------------------------------------------------------------------
516:                    cachedList := dat.([]BulkWireMsg)
--------------------------------------------------------------------------------
527:                    cachedList := dat.([]BulkWireMsg)
--------------------------------------------------------------------------------
533:            wiredMsgList := value.([]BulkWireMsg)
--------------------------------------------------------------------------------
534:            ret := strings.Split(key.(string), ":")
--------------------------------------------------------------------------------
548:            wiredMsgList := value.([]BulkWireMsg)
--------------------------------------------------------------------------------
549:            wiredMsgType := key.(string)

go-tss-master/keysign/tss_keysign.go
severity:error rule:trailofbits.go.unchecked-type-assertion.unchecked-type-assertion: Unchecked type assertion.

68:             eachParty := value.(btss.Party)

go-tss-master/messages/join_party.pb.go
severity:error rule:trailofbits.go.unchecked-type-assertion.unchecked-type-assertion: Unchecked type assertion.

264:                    switch v := v.(*JoinPartyRequest); i {
--------------------------------------------------------------------------------
276:                    switch v := v.(*JoinPartyLeaderComm); i {

go-tss-master/messages/signature_notifier.pb.go
severity:error rule:trailofbits.go.unchecked-type-assertion.unchecked-type-assertion: Unchecked type assertion.

200:                    switch v := v.(*KeysignSignature); i {
```

Gosec - Security Analysis Output Sample:

Staticcheck - Security Analysis Output Sample:

```
[redacted]
cmd/tss-recovery/tss-recovery.go:11:2: should not use dot imports (ST1001)
common/encryption_test.go:25:2: this value of encrypted is never used (SA4006)
common/encryption_test.go:27:2: this value of decrypted is never used (SA4006)
common/tss.go:101:2: field acceptedShares is unused (U1000)
common/tss.go:749:2: should merge variable declaration with assignment on next line (S1021)
common/tss_helper_test.go:107:2: this value of err is never used (SA4006)
common/tss_test.go:51:2: should merge variable declaration with assignment on next line (S1021)
common/tss_test.go:137:2: this value of err is never used (SA4006)
conversion/key_provider.go:99:2: should merge variable declaration with assignment on next line (S1021)
keygen/keygen_test.go:85:3: should merge variable declaration with assignment on next line (S1021)
keygen/keygen_test.go:98:9: unnecessary assignment to the blank identifier (S1005)
keysign/keysign_test.go:115:3: should merge variable declaration with assignment on next line (S1021)
keysign/keysign_test.go:416:9: unnecessary assignment to the blank identifier (S1005)
keysign/signature_notifier.go:9:2: package "github.com/binance-chain/tss-lib/common" is being imported more than once (ST1019)
        keysign/signature_notifier.go:10:2: other import of "github.com/binance-chain/tss-lib/common"
keysign/signature_notifier.go:11:2: package github.com/golang/protobuf/proto is deprecated: Use the "google.golang.org/protobuf/proto" package instead.  (SA1019)
keysign/tss_keysign.go:113:16: error strings should not end with punctuation or a newline (ST1005)
keysign/tss_keysign.go:171:3: should use 'return a.Cmp(b) != -1' instead of 'if a.Cmp(b) == -1 { return false }; return true' (S1008)
p2p/party_coordinator.go:89:2: redundant return statement (S1023)
p2p/stream_helper_test.go:72:2: ineffective assignment to field MockNetworkStream.protocol (SA4005)
storage/localstate_mgr_test.go:46:2: this value of fileName is never used (SA4006)
tss/keysign.go:182:2: redundant return statement (S1023)
tss/keysign.go:326:21: func (*TssServer).isPartOfKeysignParty is unused (U1000)
[redacted]
```

Ineffassign - Security Analysis Output Sample:

```
[redacted]/common/encryption_test.go:25:2: ineffectual assignment to encrypted
[redacted]/common/encryption_test.go:27:2: ineffectual assignment to decrypted
[redacted]/common/tss_helper_test.go:107:15: ineffectual assignment to err
[redacted]/common/tss_test.go:137:7: ineffectual assignment to err
[redacted]/storage/localstate_mgr_test.go:46:2: ineffectual assignment to fileName
[redacted]
```

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 on the report as security concerns.

THANK YOU FOR CHOOSING

HALBORN