// HALBORN

Pocket Network - Validator WPokt

Golang Security Assessment

Prepared by: **Halborn**

Date of Engagement: **August 21st, 2023 - August 31st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	08/21/2023
0.2	Draft Version	08/23/2023
0.3	Draft Review	08/29/2023
1.0	Remediation Plan	08/30/2023
1.1	Remediation Plan Review	08/31/2023

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Pocket Network engaged Halborn to conduct a security assessment on their validator codebase beginning on August 21st, 2023 and ending on August 31st, 2023. The security assessment was scoped to the Validator code provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned two full-time security engineers to verify the security of the merge requests. 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 assessment is to:

- Ensure that the **Validator** operates as intended.
- Identify potential security issues with the validator implementation.

In summary, Halborn identified some security risks that were successfully addressed by the Pocket Network team.

1.3 TEST APPROACH & METHODOLOGY

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

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

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

IN-SCOPE CODE & COMMIT:

Code repositories: wpokt-validator

1. ASSESSED COMMIT ID:

- Commit ID: 257a51bc6c51ef8107815870e993eb1f7e44ec78

2. ASSESSED COMMIT ID:

- Commit ID: 04950788585191126fe2e51cf091c740414e7b2b

REMEDIATION COMMIT IDs & TREE :

Remediation Tree: wpokt-validator

- Commit ID: e0d4ec8ea7bbf2c77e2c4e2aabdae081baec7fc4
- bdb5ef0531750d12ae92c170f546b1aad6fbbd3e
- 5f383e00495fd185eaf7ba4f03172613c035f95e
- 31987ccf4c9b9d19c408b97424f2bb55b6ce9de1
- bd29988867fcdfaceef27f3dcfefc62fac7b8d51
- 1a8772904d37b96958938b84d72a2d46db7c9806
- 28f3734e69fa86277339b7616ad228b2888f93c8
- 181dc4e71d4658cf5b9b0aa2c67fcfbfc65a9d20
- 0d1c02e3bcf6d72df038bfde6c8e7be80189ad55
- 781cc109b2ed77615bad524ddb8834009df801a7
- 28f3734e69fa86277339b7616ad228b2888f93c8
- 0d1c02e3bcf6d72df038bfde6c8e7be80189ad55
- a7bf68e10705cd880dad14d53277dadd586590a5

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	5	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF ON-CHAIN VERIFICATION FOR numSigners CAN LEAD TO INCORRECT STATUS UPDATE	Medium (6.3)	SOLVED - 08/30/2023
(HAL-02) LACK OF ERROR HANDLING FOR filter.Next() IN THE SyncBlocks FUNCTION	Medium (5.9)	SOLVED - 08/30/2023
(HAL-03) POTENTIAL RISK OF int64 Nonce OVERFLOW IN FINDNONCE FUNCTION OF MintSignerRunner	Medium (5.9)	SOLVED - 08/30/2023
(HAL-04) UNHANDLED ZERO ADDRESS IN ValidateMint FUNCTION CAN LEAD TO UNINTENDED BURN OPERATIONS	Low (3.8)	SOLVED - 08/30/2023
(HAL-05) DOCKER IMAGE RUNNING AS ROOT	Low (3.8)	SOLVED - 08/30/2023
(HAL-06) GO VERSION 1.18 IS UNSUPPORTED	Low (3.8)	SOLVED - 08/30/2023
(HAL-07) GO VERSIONS PRIOR TO 1.20 ARE MORE VULNERABLE TO SIDE-CHANNEL ATTACKS	Low (3.8)	SOLVED - 08/30/2023
(HAL-08) UNCHECKED ERROR RETURN VALUE FOR rand.Read in randomString FUNCTION	Low (3.1)	SOLVED - 08/30/2023
(HAL-09) INEFFICIENT STRING COMPARISON WITH strings.ToLower OR strings.ToUpper	Informational (0.0)	SOLVED - 08/30/2023
(HAL-10) UNUSED VARIABLES	Informational (0.0)	SOLVED - 08/30/2023
(HAL-11) UNNECESSARY BOOLEAN COMPARISONS	Informational (0.0)	SOLVED - 08/30/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) LACK OF ON-CHAIN VERIFICATION FOR numSigners CAN LEAD TO INCORRECT STATUS UPDATE - MEDIUM (6.3)

Description:

The existing architecture of the MintSignerRunner class, written in the Go programming language, does not incorporate an on-chain verification mechanism for the variable numSigners. The current design relies solely on the local configuration file to initialize this variable. This approach introduces a potential discrepancy between the off-chain numSigners and the on-chain validatorCount, particularly when validators are added or removed through the smart contract. Such a discrepancy can result in incorrect status updates, thereby compromising the integrity of the system.

Code Location:

/eth/signer.go#L378

Listing 1

```
...
    x := &MintSignerRunner{
        privateKey:             privateKey,
        address:                strings.ToLower(address),
        wpoktAddress:           strings.ToLower(app.Config.
↳ Ethereum.WrappedPocketAddress),
        vaultAddress:           strings.ToLower(app.Config.Pocket.
↳ VaultAddress),
        wpoktContract:          contract,
        mintControllerContract: mintControllerContract,
        numSigners:             len(app.Config.Ethereum.
↳ ValidatorAddresses),
        domain:                 domain,
        ethClient:              ethClient,
```

```
12          poktClient:             pokt.NewClient(),
13      }
14 ...
```

BVSS:

AO:A/AC:L/AX:M/C:H/I:H/A:N/D:N/Y:N/R:N/S:U (6.3)

Recommendation:

Before updating the status to models.StatusSigned, query the on-chain validatorCount to verify the numSigners.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by querying validator count through on-chain.

Commit ID: bdb5ef0531750d12ae92c170f546b1aad6fbbd3e

4.2 (HAL-02) LACK OF ERROR HANDLING FOR filter.Next() IN THE SyncBlocks FUNCTION - MEDIUM (5.9)

Description:

The SyncBlocks function in the MintExecutorRunner class uses a filter to iterate through mint events. While the function checks for errors when creating the filter, it does not check for errors that might occur during the iteration with filter.Next().

If filter.Next() encounters an error, the loop might not iterate through all the intended blocks, potentially leading to missed or incomplete data.

Code Location:

/audit/eth/executor.go#L105

Listing 2

```
func (x *MintExecutorRunner) SyncBlocks(startBlockNumber uint64, endBlockNumber uint64) bool {
    filter, err := x.wpoktContract.FilterMinted(&bind.FilterOpts{
        Start:   startBlockNumber,
        End:     &endBlockNumber,
        Context: context.Background(),
    }, []common.Address{}, []*big.Int{}, []*big.Int{})

    if err != nil {
        log.Errorln("[MINT EXECUTOR] Error while syncing mint events: ", err)
        return false
    }

    var success bool = true
    for filter.Next() {
        success = success && x.HandleMintEvent(filter.Event)
    }
```

```
17
18     return success
19 }
```

BVSS:

AO:A/AC:L/AX:M/C:H/I:M/A:N/D:N/Y:N/R:N/S:U (5.9)

Recommendation:

Consider using `filter.Error()` to check for any errors that might occur during the iteration.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by adding an error check.

Commit ID: 5f383e00495fd185eaf7ba4f03172613c035f95e

4.3 (HAL-03) POTENTIAL RISK OF int64 Nonce OVERFLOW IN FINDNONCE FUNCTION OF MintSignerRunner - MEDIUM (5.9)

Description:

The FindNonce function within the MintSignerRunner class utilizes the int64 data type for storing and manipulating nonces. While int64 is generally adequate for most applications, there are edge cases where an overflow could occur. Such an overflow is not just a theoretical concern; it poses a critical risk that could lead to transaction failures, data corruption.

The potential issue with int64 could arise when you are parsing the nonce with strconv.ParseInt(pendingMint.Data.Nonce, 10, 64). The ParseInt function will return an error if the number is too large to fit into an int64.

Code Location:

/eth/signer.go#L107

Listing 3

```
...
        if len(pendingMints) > 0 {
            var nonces []int64

            for _, pendingMint := range pendingMints {
                if pendingMint.Data != nil {
                    nonce, err := strconv.ParseInt(pendingMint.
↳ Data.Nonce, 10, 64)
                    if err != nil {
                        log.Error("[MINT SIGNER] Error converting
↳ nonce to int: ", err)
                        continue
                    }
                    nonces = append(nonces, nonce)
```

```
13                 }
14             }
15 ...
```

Proof Of Concept:

Listing 4

```
 1 package main
 2
 3 import (
 4     "fmt"
 5     "strconv"
 6 )
 7
 8 func FindNonce(mintNonce string) (int64, error) {
 9     var nonce int64
10     if mintNonce != "" {
11         var err error
12         nonce, err = strconv.ParseInt(mintNonce, 10, 64)
13         if err != nil {
14             return 0, err
15         }
16     }
17
18     // Simulate fetching from contract
19     currentNonce := int64(9223372036854775800) // Close to int64
↳ max value
20
21     nonce = currentNonce + 150
22     return nonce, nil
23 }
24
25 func main() {
26     fmt.Println("Testing FindNonce...")
27     findNonce, err := FindNonce("")
28     if err != nil {
29         fmt.Println("Error:", err)
30     } else {
31         fmt.Println("Returned Nonce:", findNonce)
32     }
```

Output:

Listing 5

```
1 Testing FindNonce.
2 Returned Nonce: -9223372036854775666
```

BVSS:

AO:A/AC:L/AX:M/C:H/I:M/A:N/D:N/Y:N/R:N/S:U (5.9)

Recommendation:

Consider switching to Arbitrary-Precision Arithmetic. Transition to using `big.Int` for all nonce operations to avoid any overflow issues.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by using `big.Int` for nonce.

Commit ID: 31987ccf4c9b9d19c408b97424f2bb55b6ce9de1

4.4 (HAL-04) UNHANDLED ZERO ADDRESS IN ValidateMint FUNCTION CAN LEAD TO UNINTENDED BURN OPERATIONS - LOW (3.8)

Description:

The ValidateMint function in the MintSignerRunner class is responsible for validating various aspects of a minting transaction, including the recipient address. However, the function does not currently distinguish between a missing recipient address and a zero address (0x0000000000000000000000000000000000000000). In the context of many blockchain protocols, sending tokens to a zero address is equivalent to burning them, removing them from circulation. The absence of this check could lead to unintended burn operations.

Code Location:

Listing 6

```
func (x *MintSignerRunner) ValidateMint(mint *models.Mint, data *
↳ autogen.MintControllerMintData) (bool, error) {
    log.Debug("[MINT SIGNER] Validating mint: ", mint.
↳ TransactionHash)

    tx, err := x.poktClient.GetTx(mint.TransactionHash)
    if err != nil {
        return false, errors.New("Error fetching transaction: " +
↳ err.Error())
    }

    if tx.Tx == "" || tx.TxResult.Code != 0 {
        log.Debug("[MINT SIGNER] Transaction not found or failed")
        return false, nil
    }

    if tx.TxResult.MessageType != "send" || tx.StdTx.Msg.Type != "
↳ pos/Send" {
```

```
15          log.Debug("[MINT SIGNER] Transaction message type is not
↳ send")
16          return false, nil
17      }
18
19      if strings.ToLower(tx.StdTx.Msg.Value.ToAddress) != strings.
↳ ToLower(x.vaultAddress) {
20          log.Debug("[MINT SIGNER] Transaction recipient is not
↳ vault address")
21          return false, nil
22      }
23
24      if strings.ToLower(tx.StdTx.Msg.Value.FromAddress) != strings.
↳ ToLower(mint.SenderAddress) {
25          log.Debug("[MINT SIGNER] Transaction signer is not sender
↳ address")
26          return false, nil
27      }
28
29      if tx.StdTx.Msg.Value.Amount != mint.Amount {
30          log.Debug("[MINT SIGNER] Transaction amount does not match
↳  mint amount")
31          return false, nil
32      }
33
34      memo, valid := poktUtil.ValidateMemo(tx.StdTx.Memo)
35      if !valid {
36          log.Debug("[MINT SIGNER] Memo failed validation")
37          return false, nil
38      }
39
40      if strings.ToLower(memo.Address) != strings.ToLower(mint.
↳ RecipientAddress) {
41          log.Debug("[MINT SIGNER] Memo address does not match
↳ recipient address")
42          return false, nil
43      }
44
45      if memo.ChainId != mint.RecipientChainId {
46          log.Debug("[MINT SIGNER] Memo chain id does not match
↳ recipient chain id")
47          return false, nil
48      }
49
```

```
50      log.Debug("[MINT SIGNER] Mint validated")
51      return true, nil
52  }
```

BVSS:

AO:A/AC:L/AX:M/C:M/I:L/A:N/D:N/Y:N/R:N/S:U (3.8)

Recommendation:

Implement an explicit check for the zero address in the ValidateMint function. If a zero address is detected, the function should return an error or a boolean flag indicating that the transaction is invalid for the intended operation. This will prevent unintended burn operations and improve the robustness of the minting process.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by adding a check on the zero address.

Commit ID: 28f3734e69fa86277339b7616ad228b2888f93c8

4.5 (HAL-05) DOCKER IMAGE RUNNING AS ROOT - LOW (3.8)

Description:

Docker containers generally run with root privileges by default. This allows for unrestricted container management, meaning a user could install system packages, edit configuration files, bind privileged ports, etc. During static analysis, it was observed that the docker image is maintained through the root user.

Code Location:

Listing 7: Dockerfile

```
1  FROM golang:1.19 as base
2
3  WORKDIR /app
4
5  COPY go.mod go.sum ./
6  RUN go mod download
7
8  # copy the source code
9  COPY app ./app
10 COPY eth ./eth
11 COPY pokt ./pokt
12 COPY models ./models
13 COPY main.go ./
14 COPY defaults.yml ./
15
16 # build
17 RUN CGO_ENABLED=0 GOOS=linux go build -o /validator
18
19 # set environment variables
20 # mongodb
21 ENV MONGODB_URI ${MONGODB_URI}
22 ENV MONGODB_DATABASE ${MONGODB_DATABASE}
23 ENV MONGODB_TIMEOUT_SECS ${MONGODB_TIMEOUT_SECS}
24
25 # ethereum
```

```
26 ENV ETH_PRIVATE_KEY ${ETH_PRIVATE_KEY}
27 ENV ETH_RPC_URL ${ETH_RPC_URL}
28 ENV ETH_CHAIN_ID ${ETH_CHAIN_ID}
29 ENV ETH_START_BLOCK_NUMBER ${ETH_START_BLOCK_NUMBER}
30 ENV ETH_CONFIRMATIONS ${ETH_CONFIRMATIONS}
31 ENV ETH_RPC_TIMEOUT_SECS ${ETH_RPC_TIMEOUT_SECS}
32 ENV ETH_WRAPPED_POCKET_ADDRESS ${ETH_WRAPPED_POCKET_ADDRESS}
33 ENV ETH_MINT_CONTROLLER_ADDRESS ${ETH_MINT_CONTROLLER_ADDRESS}
34 ENV ETH_VALIDATOR_ADDRESSES ${ETH_VALIDATOR_ADDRESSES}
35
36 # pocket
37 ENV POKT_PRIVATE_KEY ${POKT_PRIVATE_KEY}
38 ENV POKT_RPC_URL ${POKT_RPC_URL}
39 ENV POKT_CHAIN_ID ${POKT_CHAIN_ID}
40 ENV POKT_START_HEIGHT ${POKT_START_HEIGHT}
41 ENV POKT_CONFIRMATIONS ${POKT_CONFIRMATIONS}
42 ENV POKT_RPC_TIMEOUT_SECS ${POKT_RPC_TIMEOUT_SECS}
43 ENV POKT_TX_FEE ${POKT_TX_FEE}
44 ENV POKT_VAULT_ADDRESS ${POKT_VAULT_ADDRESS}
45 ENV POKT_MULTISIG_PUBLIC_KEYS ${POKT_MULTISIG_PUBLIC_KEYS}
46
47 # google secret manager
48 ENV GOOGLE_SECRET_MANAGER_ENABLED ${GOOGLE_SECRET_MANAGER_ENABLED}
49 ENV GOOGLE_MONGO_SECRET_NAME ${GOOGLE_MONGO_SECRET_NAME}
50 ENV GOOGLE_POKT_SECRET_NAME ${GOOGLE_POKT_SECRET_NAME}
51 ENV GOOGLE_ETH_SECRET_NAME ${GOOGLE_ETH_SECRET_NAME}
52
53 # mint monitor
54 ENV MINT_MONITOR_ENABLED ${MINT_MONITOR_ENABLED}
55 ENV MINT_MONITOR_INTERVAL_SECS ${MINT_MONITOR_INTERVAL_SECS}
56
57 # mint signer
58 ENV MINT_SIGNER_ENABLED ${MINT_SIGNER_ENABLED}
59 ENV MINT_SIGNER_INTERVAL_SECS ${MINT_SIGNER_INTERVAL_SECS}
60
61 # mint executor
62 ENV MINT_EXECUTOR_ENABLED ${MINT_EXECUTOR_ENABLED}
63 ENV MINT_EXECUTOR_INTERVAL_SECS ${MINT_EXECUTOR_INTERVAL_SECS}
64
65 # burn monitor
66 ENV BURN_MONITOR_ENABLED ${BURN_MONITOR_ENABLED}
67 ENV BURN_MONITOR_INTERVAL_SECS ${BURN_MONITOR_INTERVAL_SECS}
68
69 # burn signer
```

```
70 ENV BURN_SIGNER_ENABLED ${BURN_SIGNER_ENABLED}
71 ENV BURN_SIGNER_INTERVAL_SECS ${BURN_SIGNER_INTERVAL_SECS}
72
73 # burn executor
74 ENV BURN_EXECUTOR_ENABLED ${BURN_EXECUTOR_ENABLED}
75 ENV BURN_EXECUTOR_INTERVAL_SECS ${BURN_EXECUTOR_INTERVAL_SECS}
76
77 # health check
78 ENV HEALTH_CHECK_INTERVAL_SECS ${HEALTH_CHECK_INTERVAL_SECS}
79
80 # logging
81 ENV LOG_LEVEL ${LOG_LEVEL}
82
83 # run
84 CMD ["/validator", "--config", "/app/defaults.yml"]
```

BVSS:

AO:A/AC:L/AX:M/C:M/I:L/A:N/D:N/Y:N/R:N/S:U (3.8)

Recommendation:

It is recommended to build the Dockerfile and run the container as a non-root user.

Listing 8: Reference

```
1 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.
```

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by running the container as a non-root user.

Commit ID: 181dc4e71d4658cf5b9b0aa2c67fcfbfc65a9d20

4.6 (HAL-06) GO VERSION 1.18 IS UNSUPPORTED - LOW (3.8)

Description:

The project uses Go version 1.18. This version has been deprecated. See the Go release notes for their policy on supporting major versions of Go.

Code Location:

go.mod

Listing 9

```
1 module github.com/dan13ram/wpokt-validator
2
3 go 1.18
```

BVSS:

AO:A/AC:L/AX:M/C:M/I:L/A:N/D:N/Y:N/R:N/S:U (3.8)

Recommendation:

Update to a supported version of Go in order to receive ongoing security updates.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by updating the go version.

Commit ID: 1a8772904d37b96958938b84d72a2d46db7c9806

4.7 (HAL-07) GO VERSIONS PRIOR TO 1.20 ARE MORE VULNERABLE TO SIDE-CHANNEL ATTACKS - LOW (3.8)

Description:

Go version 1.20.2 contains security and performance enhancements. Specifically, this release fixes problems in cryptographic libraries. Older versions of go are more susceptible to cryptography issues and side-channel attacks on cryptographic implementations.

Code Location:

go.mod

Listing 10

```
1 module github.com/dan13ram/wpokt-validator
2
3 go 1.18
```

BVSS:

AO:A/AC:L/AX:M/C:M/I:L/A:N/D:N/Y:N/R:N/S:U (3.8)

Recommendation:

Update to at least Go v1.20.2.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by updating the go version.

Commit ID: 1a8772904d37b96958938b84d72a2d46db7c9806

4.8 (HAL-08) UNCHECKED ERROR RETURN VALUE FOR rand.Read in randomString FUNCTION - LOW (3.1)

Description:

The randomString function in app/database.go uses the rand.Read function to generate a random byte slice. However, the error return value of rand.Read is not checked, which could lead to unpredictable behavior or insecure random strings if the function fails to read random bytes.

Code Location:

Listing 11

```
app/database.go:83:11: Error return value of `rand.Read` is not
↳ checked (errcheck)
    rand.Read(bytes)
```

BVSS:

AO:A/AC:L/AX:M/C:L/I:L/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation:

Add error handling to check the return value of rand.Read. If an error occurs, handle it appropriately, such as logging the error and returning an empty string or propagating the error up to the caller.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by adding error handling.

Commit ID: 0d1c02e3bcf6d72df038bfde6c8e7be80189ad55

4.9 (HAL-09) INEFFICIENT STRING COMPARISON WITH strings.ToLower OR strings.ToUpper - INFORMATIONAL (0.0)

Description:

Converting two strings to the same case and comparing them like so :

Listing 12

```
if strings.ToLower(s1) == strings.ToLower(s2) {
    ...
}
```

is significantly more expensive than comparing them with strings.EqualFold (s1, s2). This is due to memory usage as well as computational complexity.

strings.ToLower will have to allocate memory for the new strings, as well as convert both strings fully, even if they differ on the very first byte. strings.EqualFold, on the other hand, compares the strings one character at a time. It doesn't need to create two intermediate strings, and can return as soon as the first non-matching character has been found.

Code Location:

Listing 13

```
eth/signer.go:156:5:
    if strings.ToLower(tx.StdTx.Msg.Value.ToAddress) != strings.
↳ ToLower(x.vaultAddress) {
eth/signer.go:161:5:
    if strings.ToLower(tx.StdTx.Msg.Value.FromAddress) != strings.
↳ ToLower(mint.SenderAddress) {
eth/signer.go:177:5:
    if strings.ToLower(memo.Address) != strings.ToLower(mint.
```

```
↳ RecipientAddress) {
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

Consider using strings.EqualFold instead toLower comparison.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by using strings.EqualFold.

Commit ID: 781cc109b2ed77615bad524ddb8834009df801a7

4.10 (HAL-10) UNUSED VARIABLES - INFORMATIONAL (0.0)

Description:

The util package contains a variable named passphrase, which is initialized but never used within the package.

Code Location:

util/signer.go#L25

Listing 14

```
pokt/util/signer.go:20:7:
const legacyCodec bool = false
      ^
pokt/util/signer.go:25:5:
var passphrase string = uniuri.NewLen(32)
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

Consider deleting unused variables.

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by deleting unused variables.

Commit ID: a7bf68e10705cd880dad14d53277dadd586590a5

4.11 (HAL-11) UNNECESSARY BOOLEAN COMPARISONS - INFORMATIONAL (0.0)

Description:

The Go codebase contains several instances where boolean variables are unnecessarily compared to boolean constants (true or false).

Code Location:

Listing 15

```
eth/executor.go:147:5:
`!app.Config.MintExecutor.Enabled` (gosimple)
    if app.Config.MintExecutor.Enabled == false {
       ^
eth/monitor.go:122:5:
`!app.Config.BurnMonitor.Enabled` (gosimple)
    if app.Config.BurnMonitor.Enabled == false {
       ^
eth/signer.go:327:5:
`!app.Config.MintSigner.Enabled` (gosimple)
    if app.Config.MintSigner.Enabled == false {

pokt/signer.go:355:5: `!app.Config.BurnSigner.Enabled` (gosimple)
    if app.Config.BurnSigner.Enabled == false {
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

Consider removing the unnecessary comparison to boolean constants.

- Before: `if app.Config.MintExecutor.Enabled == false {`

- After: `if !app.Config.MintExecutor.Enabled {`
- Before: `if app.Config.BurnMonitor.Enabled == false {`
- After: `if !app.Config.BurnMonitor.Enabled {`
- Before: `if app.Config.MintSigner.Enabled == false {`
- After: `if !app.Config.MintSigner.Enabled {`

Remediation Plan:

SOLVED: The Pocket Network team resolved the issue by deleting redundant comparison.

Commit ID: bd29988867fcdfaceef27f3dcfefc62fac7b8d51

AUTOMATED TESTING

5.1 Description

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

5.2 Semgrep

Security Analysis Output Sample:

Listing 16: Rule Set

```
semgrep --config "p/dgryski.semgrep-go" x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o dgryski.semgrep
semgrep --config "p/owasp-top-ten"      x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o owasp-top-ten.semgrep
semgrep --config "p/r2c-security-audit" x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o r2c-security-audit.semgrep
semgrep --config "p/r2c-ci"             x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o r2c-ci.semgrep
semgrep --config "p/ci"                 x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o ci.semgrep
semgrep --config "p/golang"             x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o golang.semgrep
semgrep --config "p/trailofbits"        x/liquidstakeibc --exclude='*_test.go' --max-lines-per-finding 1000 --no-git-ignore -o trailofbits.semgrep
```

Semgrep Results:

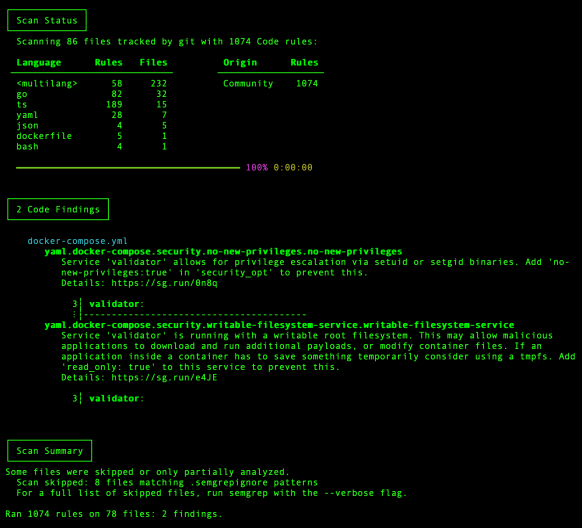

Figure 1: Semgrep results

- No major issues found by Semgrep.

5.3 Gosec

Analysis Output Sample:

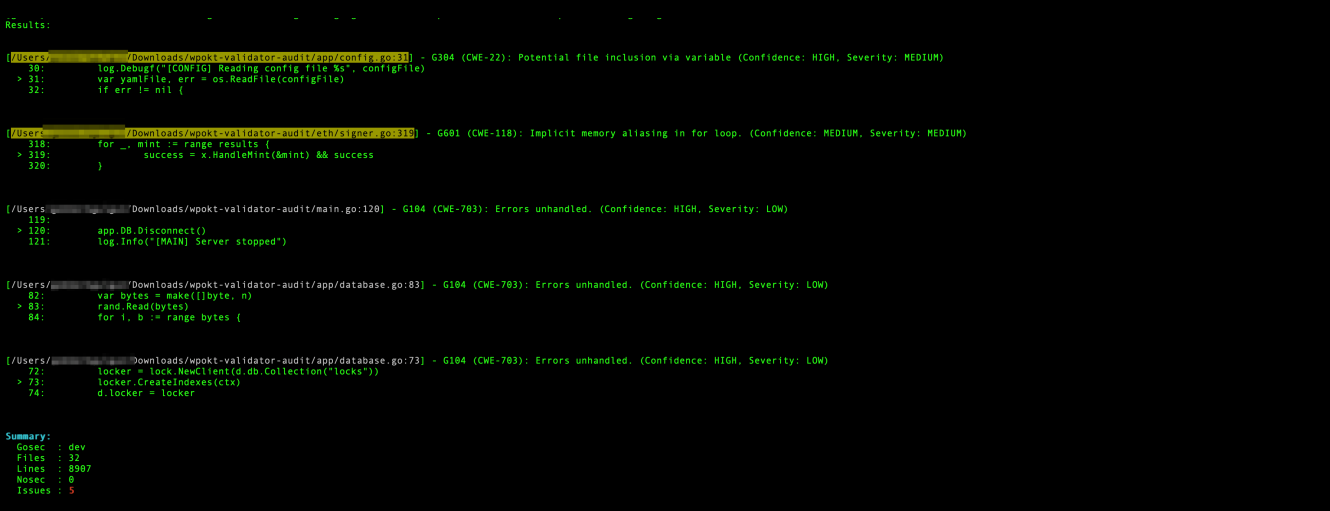

Figure 2: Gosec results

- No major issues found by Gosec.

5.4 Golang Lint

Analysis Output Sample:

```
app/database.go:73:22: Error return value of `locker.CreateIndexes` is not checked (errcheck)
        locker.CreateIndexes(ctx)
                           ^
app/database.go:83:11: Error return value of `rand.Read` is not checked (errcheck)
        rand.Read(bytes)
                 ^
main.go:120:19: Error return value of `app.DB.Disconnect` is not checked (errcheck)
        app.DB.Disconnect()
                         ^
pokt/client/pokt_client.go:36:2: var `getNodePath` is unused (unused)
        getNodePath,
        ^
pokt/client/pokt_client.go:37:2: var `getACLPath` is unused (unused)
        getACLPath,
        ^
pokt/client/pokt_client.go:38:2: var `getUpgradePath` is unused (unused)
        getUpgradePath,
        ^
pokt/client/pokt_client.go:39:2: var `getDAOOwnerPath` is unused (unused)
        getDAOOwnerPath,
        ^
pokt/client/pokt_client.go:41:2: var `getAccountPath` is unused (unused)
        getAccountPath,
        ^
pokt/client/pokt_client.go:42:2: var `getAppPath` is unused (unused)
        getAppPath,
        ^
pokt/client/pokt_client.go:45:2: var `getSupportedChainsPath` is unused (unused)
        getSupportedChainsPath,
        ^
pokt/client/pokt_client.go:46:2: var `getBalancePath` is unused (unused)
        getBalancePath,
        ^
pokt/client/pokt_client.go:48:2: var `getNodeParamsPath` is unused (unused)
        getNodeParamsPath,
        ^
pokt/client/pokt_client.go:49:2: var `getNodesPath` is unused (unused)
        getNodesPath,
        ^
pokt/client/pokt_client.go:50:2: var `getSigningInfoPath` is unused (unused)
        getSigningInfoPath,
        ^
pokt/client/pokt_client.go:51:2: var `getAppsPath` is unused (unused)
        getAppsPath,
        ^
pokt/client/pokt_client.go:52:2: var `getAppParamsPath` is unused (unused)
        getAppParamsPath,
        ^
pokt/client/pokt_client.go:53:2: var `getPocketParamsPath` is unused (unused)
        getPocketParamsPath,
        ^
pokt/client/pokt_client.go:54:2: var `getNodeClaimsPath` is unused (unused)
        getNodeClaimsPath,
        ^
pokt/client/pokt_client.go:55:2: var `getNodeClaimPath` is unused (unused)
        getNodeClaimPath,
        ^
pokt/client/pokt_client.go:56:2: var `getBlockTxsPath` is unused (unused)
        getBlockTxsPath,
        ^
pokt/client/pokt_client.go:57:2: var `getSupplyPath` is unused (unused)
```

Figure 3: Golang Lint results

- No major issues found by Golang Lint results.

THANK YOU FOR CHOOSING

// HALBORN