Blockchain Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.11.28, the SlowMist security team received the Blox Staking team's security audit application for ssv-dkg, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

In black box testing and gray box testing, we use methods such as fuzz testing and script testing to test the robustness of the interface or the stability of the components by feeding random data or constructing data with a specific structure, and to mine some boundaries Abnormal performance of the system under conditions such as bugs or abnormal performance. In white box testing, we use methods such as code review, combined with the relevant experience accumulated by the security team on known blockchain security vulnerabilities, to analyze the object definition and logic implementation of the code to ensure that the code has the key components of the key logic. Realize no known vulnerabilities; at the same time, enter the vulnerability mining mode for new scenarios and new technologies, and find possible 0day errors.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

NO.	Audit Items	Result
1	Design Logic Audit	Some Risks
2	Others	Some Risks
3	State Consistency Audit	Some Risks
4	Failure Rollback Audit	Passed
5	Unit Test Audit	Passed
6	Integer Overflow Audit	Passed
7	Parameter Verification Audit	Some Risks
8	Error Unhandle Audit	Some Risks

NO.	Audit Items	Result
9	Boundary Check Audit	Passed
10	SAST	Some Risks

3 Project Overview

3.1 Project Introduction

The SSV team built this tool leveraging drand's DKG protocol implementation. This implementation operates under the assumption of a p2p network, allowing operators to communicate. The ssv-dkg was built to lift this assumption and provide a communication layer that centered on an Initiator figure, to facilitate communication between operators. The introduced potential risk for centralization and bad actors is handled with signatures and signature verifications, as explained in the Security notes section. Finally, the outcome of the DKG ceremony is a BLS key pair to be used for validator duties by operators on the ssv.network. As such, the tool ends the process by creating a deposit file to activate the newly created validator key pair, and proceeds to generating the payload for the transaction.

3.2 Coverage

Target Code and Revision:

https://github.com/bloxapp/ssv-dkg

Initial audit commit: e4acbffd35879765b4dd36bd029410d46c5ef10b

Patch review commit: ee49ca98ddce157f892e7728c9339dcde8b70f28

Final review commit: 000d41b4f1834368e5e1a58e9c1a50374995d50c

3.3 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	RSA keys should be at least 2048 bits	SAST	Low	Fixed
N2	File permissions 0o644 too more	SAST	Information	Fixed
N3	Errors unhandled of `postF` function	Error Unhandle Audit	Low	Fixed
N4	Directory traversal attack risk	Parameter Verification Audit	Information	Fixed
N5	Using fmt.Printf for logging in production	Others	Information	Fixed
N6	Unhandled case in `LoadRSAPrivKey` function	Design Logic Audit	Medium	Fixed
N7	Missing `nil` check when calling `pem.Decode` function	Design Logic Audit	Medium	Fixed
N8	Potential `nil` return when calling `pem.EncodeToMemory`	Design Logic Audit	Medium	Fixed
N9	Non-deterministic order in BLS signature recovery	State Consistency Audit	Critical	Fixed
N10	Potential looping issue in `operatorID`	Design Logic Audit	Low	Fixed
N11	`x509.IsEncryptedPEMBlock/x509.DecryptPEMBlock` are deprecated	Others	Medium	Fixed
N12	Logging of sensitive information	Others	Information	Fixed
N13	Delayed handling of errors	Error Unhandle Audit	Medium	Fixed
N14	Improper use of panic in error handling	Error Unhandle Audit	High	Fixed
N15	Potential slowloris attack due to missing	Others	Medium	Fixed

NO	Title	Category	Level	Status
	ReadHeaderTimeout configuration			
N16	Lack of system time accuracy check on service startup	Others	Low	Acknowledged
N17	`drand_bls` library is deprecated	Others	Low	Fixed
N18	Lack of Handling Unresponsive Hosts	Design Logic Audit	Medium	Fixed
N19	Lack of Handling All Switch Cases and Missing Default Case	Design Logic Audit	Medium	Fixed
N20	Mixed Value and Pointer Receivers	Others	Information	Fixed
N21	Lack of HTTPS Usage	Design Logic Audit	Low	Fixed

4 Findings

4.1 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

4.2 Vulnerability Summary

[N1] [Low] RSA keys should be at least 2048 bits

Category: SAST

Content

- pkgs/crypto/crypto.go

```
func GenerateKeys() (*rsa.PrivateKey, *rsa.PublicKey, error) {
        pv, err := rsa.GenerateKey(rand.Reader, 1024)
        if err != nil {
                return nil, nil, err
```

```
        }

        return pv, &pv.PublicKey, nil

}
```

Here's a rough guideline for RSA key lengths and their security levels:

```
1024 bits: Insecure, not recommended for most purposes.
2048 bits: Standard for most uses and considered secure.
3072 bits or higher: Provides extra security, especially for sensitive applications.
```

As technology advances, the computing power required to break encryption also increases, and longer key lengths become necessary to maintain a similar level of security.

Solution

It is generally recommended to use a key length of 2048 bits or higher for RSA in contemporary applications.

Status

Fixed

[N2] [Information] File permissions 0o644 too more

Category: SAST

Content

- cli/utils/utils.go

```
    87:         // If the log file doesn't exist, create it
  > 88:         _, err := os.OpenFile(LogFilePath,
os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0o644)
    89:         if err != nil {
```

- cli/initiator/initiator.go

```
    94:                                 }
  > 95:                                 err = os.WriteFile(rsaKeyPasswordPath,
[]byte(password), 0o644)
    96:                                 if err != nil {
```

- cli/initiator/initiator.go

```
    84:                             rsaKeyPath := fmt.Sprintf("%s/encrypted_private_key-
%v.json", cli_utils.OutputPath, depositData.PubKey)
  > 85:                             err = os.WriteFile(rsaKeyPath, encryptedRSAJSON,
0o644)
    86:                             if err != nil {
```

In Unix-like systems, file permissions are specified using three octal digits, where each digit represents the permission set for the owner, group, and others, respectively.

- `0o644`: The octal prefix `0o` indicates that the number following is in octal format. In this case, `0644` means read and write permissions for the owner and read permissions for others.
- `0600`: This is more restrictive, allowing only read and write permissions for the owner and no permissions for others.

The recommendation here is to set the permissions to 0600 to limit access to the file to the owner only. This is often advisable for sensitive information, such as passwords, to prevent unauthorized access.

Solution

By setting the permissions to 0600, you ensure that only the owner has read and write access to the file, enhancing the security of sensitive information like passwords.

Status

Fixed

[N3] [Low] Errors unhandled of `postF` function

Category: Error Unhandle Audit

Content

- pkgs/dkg/drand.go

```
    309:                 res := <-p.WaitEnd()
  > 310:                 postF(&res)
    311:         }(p, o.postReshare)
...
    242:                 res := <-p.WaitEnd()
  > 243:                 postF(&res)
    244:         }(p, o.postReshare)
...
    206:                 res := <-p.WaitEnd()
```

```
> 207:                 postF(&res)
  208:         }(p, o.PostDKG)
```

If the `postF` function returns an error, but the error is not checked or handled in code, it could lead to potential security or reliability issues.

Solution

Handle the error appropriately, such as logging or returning an error.

Status

Fixed

[N4] [Information] Directory traversal attack risk

Category: Parameter Verification Audit

Content

- cli/flags/flags.go

```
// OperatorsInfoFlag  adds path to where to look for operator info file flag to the command
func OperatorsInfoPathFlag(c *cobra.Command) {
        AddPersistentStringFlag(c, operatorsInfoPath, "", "Path to where to look for operator info file", false)
}

// GetOperatorsInfoPathFlagValue gets path to where to look for operator info file flag from the command
func GetOperatorsInfoPathFlagValue(c *cobra.Command) (string, error) {
        return c.Flags().GetString(operatorsInfoPath)
}

// OperatorConfigPathFlag config path flag to the command
func ConfigPathFlag(c *cobra.Command) {
        AddPersistentStringFlag(c, configPath, "", "Path to config file", false)
}

// GetConfigPathFlagValue gets config path flag from the command
func GetConfigPathFlagValue(c *cobra.Command) (string, error) {
        return c.Flags().GetString(configPath)
}
// LogFilePathFlag file path to write logs into
func LogFilePathFlag(c *cobra.Command) {
        AddPersistentStringFlag(c, logFilePath, "./data/debug.log", "Defines a file
```

```
path to write logs into", false)
}

// GetLogFilePathValue gets logs file path flag from the command
func GetLogFilePathValue(c *cobra.Command) (string, error) {
        return c.Flags().GetString(logFilePath)
}

// DBPathFlag adds path for storage flag to the command
func DBPathFlag(c *cobra.Command) {
        AddPersistentStringFlag(c, DBPath, "./data/db", "Path for storage", false)
}

// GetDBPathFlagValue gets path for storage flag from the command
func GetDBPathFlagValue(c *cobra.Command) (string, error) {
        return c.Flags().GetString(DBPath)
}

func ResultPathFlag(c *cobra.Command) {
        AddPersistentStringFlag(c, outputPath, "./", "Path to store results", false)
}

func GetResultPathFlag(c *cobra.Command) (string, error) {
        return c.Flags().GetString(outputPath)
}
```

A directory traversal attack (also known as path traversal or ../ attack) occurs when an attacker is able to navigate up the directory structure and access files or directories outside of the intended directory. To mitigate this risk, you can perform input validation on the outputPath to ensure it doesn't contain sequences like "../".

Solution

Checks if the outputPath contains "../". If it does, an error is returned.

Status

Fixed

[N5] [Information] Using fmt.Printf for logging in production

Category: Others

Content

- cli/utils/utils.go

```
  80,3:                   fmt.Println("⚠️ config file was not provided, using flag
parameters")
  106,3:                  fmt.Println("🔑 path to password file is provided -
decrypting")
  124,3:                  fmt.Println("🔑 password for key NOT provided - trying to
read plaintext key")
  143,3:                  fmt.Println("🔑 path to password file is provided")
  196,3:                  fmt.Println("📖 reading operators info JSON file")
  282,3:                  fmt.Println("⚠️ debug log path was not provided, using
default: ./initiator_debug.log")
```

Using `fmt.Printf` for logging in a production environment may result in performance issues. While `fmt.Printf` is a flexible formatting function suitable for development and debugging information, it may not be optimized for high-performance logging.

Solution

In a production setting, it is generally recommended to use specialized logging libraries like `logrus`, `zap`, etc., which are designed for efficient and feature-rich logging.

Status

Fixed

[N6] [Medium] Unhandled case in `LoadRSAPrivKey` function

Category: Design Logic Audit

Content

- cli/utils/utils.go

```
func LoadRSAPrivKey() (*rsa.PrivateKey, []byte, error) {
        var privateKey *rsa.PrivateKey
        var encryptedRSAJSON []byte
        var err error
        if PrivKey != "" && !GenerateInitiatorKey {
                privateKey, err = OpenPrivateKey(PrivKeyPassword, PrivKey)
                if err != nil {
                        return nil, nil, err
                }
        }
        if PrivKey == "" && GenerateInitiatorKey {
                privateKey, encryptedRSAJSON, err =
GenerateRSAKeyPair(PrivKeyPassword, PrivKey)
```

```
                if err != nil {
                        return nil, nil, err
                }
        }
        return privateKey, encryptedRSAJSON, nil
}
```

The `LoadRSAPrivKey` function does not handle the case where `PrivKey` is neither empty nor set to generate an initiator key (`GenerateInitiatorKey` is false). The function currently checks two conditions independently, and if neither condition is met, it falls through without explicitly handling the case.

Solution

Ensure that the function covers all possible cases for the `PrivKey` variable. Add an explicit `else` condition to handle the case where `PrivKey` is neither empty nor set to generate an initiator key.

Status

Fixed

[N7] [Medium] Missing `nil` check when calling `pem.Decode` function

Category: Design Logic Audit

Content

- pkgs/crypto/crypto.go

```
func ParseRSAPubkey(pk []byte) (*rsa.PublicKey, error) {
        operatorKeyByte, err := base64.StdEncoding.DecodeString(string(pk))
        if err != nil {
                return nil, err
        }
        pemblock, _ := pem.Decode(operatorKeyByte)) //SlowMist//
        pbkey, err := x509.ParsePKIXPublicKey(pemblock.Bytes)
        if err != nil {
                return nil, err
        }
        return pbkey.(*rsa.PublicKey), nil
}

func PrivateKey(path string) (*rsa.PrivateKey, error) {
        data, err := os.ReadFile(path)
        if err != nil {
                return nil, fmt.Errorf("failed to read file: %w", err)
        }
```

```
        operatorKeyByte, err := base64.StdEncoding.DecodeString(string(data))
        if err != nil {
                return nil, err
        }
        block, _ := pem.Decode(operatorKeyByte) //SlowMist//
        // TODO: resolve deprecation https://github.com/golang/go/issues/8860
        enc := x509.IsEncryptedPEMBlock(block) //nolint
        b := block.Bytes
        if enc {
                var err error
                // TODO: resolve deprecation https://github.com/golang/go/issues/8860
                b, err = x509.DecryptPEMBlock(block, nil) //nolint
                if err != nil {
                        return nil, err
                }
        }
        parsedSk, err := x509.ParsePKCS1PrivateKey(b)
        if err != nil {
                return nil, err
        }
        return parsedSk, nil
}
```

The `ParseRSAPubkey` function lacks a nil check after calling `pem.Decode(operatorKeyByte)`. If the decoding process fails to find PEM data, `pemblock` will be `nil`, and the function proceeds without handling this scenario. This omission may lead to a nil pointer dereference error in subsequent code.

Solution

Ensure that `pemblock` is not `nil` after calling `pem.Decode(operatorKeyByte)`. If `pemblock` is `nil`, return an appropriate error to handle the case where no PEM data is found in the input.

Status

Fixed

[N8] [Medium] Potential `nil` return when calling `pem.EncodeToMemory`

Category: Design Logic Audit

Content

- pkgs/crypto/crypto.go

```
func EncodePublicKey(pk *rsa.PublicKey) ([]byte, error) {
        pkBytes, err := x509.MarshalPKIXPublicKey(pk)
        if err != nil {
                return nil, err
        }
        pemByte := pem.EncodeToMemory( //SlowMist//
                &pem.Block{
                        Type:  "RSA PUBLIC KEY",
                        Bytes: pkBytes,
                },
        )

        return []byte(base64.StdEncoding.EncodeToString(pemByte)), nil
}

func ExtractPrivateKey(sk *rsa.PrivateKey) string {
        pemByte := pem.EncodeToMemory( //SlowMist//
                &pem.Block{
                        Type:  "RSA PRIVATE KEY",
                        Bytes: x509.MarshalPKCS1PrivateKey(sk),
                },
        )
        return base64.StdEncoding.EncodeToString(pemByte)
}
```

The `EncodePublicKey` function uses `pem.EncodeToMemory` to encode the RSA public key to PEM format. However, the `pem.EncodeToMemory` function may return `nil` if an error occurs during encoding. The function currently does not check for the possibility of a `nil` return from `pem.EncodeToMemory`, which could lead to unexpected behavior or nil pointer dereference issues.

Solution

Check whether the result of `pem.EncodeToMemory` is `nil` and handle this condition by returning an appropriate error.

Status

Fixed

[N9] [Critical] Non-deterministic order in BLS signature recovery

Category: State Consistency Audit

Content

- pkgs/crypto/crypto.go

```
// RecoverValidatorPublicKey recovers a BLS master public key (validator pub key) from
provided partial pub keys
func RecoverValidatorPublicKey(sharePks map[uint64]*bls.PublicKey) (*bls.PublicKey,
error) {
        validatorRecoveredPK := bls.PublicKey{}
        idVec := make([]bls.ID, 0)
        pkVec := make([]bls.PublicKey, 0)
        for index, pk := range sharePks {
                blsID := bls.ID{}
                if err := blsID.SetDecString(fmt.Sprintf("%d", index)); err != nil {
                        return nil, err
                }
                idVec = append(idVec, blsID)
                pkVec = append(pkVec, *pk)
        }
        if err := validatorRecoveredPK.Recover(pkVec, idVec); err != nil {
//SlowMist//
                return nil, fmt.Errorf("error recovering validator pub key from
shares")
        }
        return &validatorRecoveredPK, nil
}

// RecoverMasterSig recovers a BLS master signature from T-threshold partial
signatures
func RecoverMasterSig(sigDepositShares map[uint64]*bls.Sign) (*bls.Sign, error) {
        reconstructedDepositMasterSig := bls.Sign{}
        idVec := make([]bls.ID, 0)
        sigVec := make([]bls.Sign, 0)
        for index, sig := range sigDepositShares {
                blsID := bls.ID{}
                if err := blsID.SetDecString(fmt.Sprintf("%d", index)); err != nil {
                        return nil, err
                }
                idVec = append(idVec, blsID)
                sigVec = append(sigVec, *sig)
        }
        if err := reconstructedDepositMasterSig.Recover(sigVec, idVec); err != nil {
//SlowMist//
                return nil, fmt.Errorf("deposit root signature recovered from shares
is invalid")
        }
        return &reconstructedDepositMasterSig, nil
}
// ReconstructSignatures receives a map of user indexes and serialized bls.Sign.
// It then reconstructs the original threshold signature using lagrange interpolation
```

```
func ReconstructSignatures(signatures map[uint64][]byte) (*bls.Sign, error) {
        reconstructedSig := bls.Sign{}
        idVec := make([]bls.ID, 0)
        sigVec := make([]bls.Sign, 0)
        for index, signature := range signatures {
                blsID := bls.ID{}
                err := blsID.SetDecString(fmt.Sprintf("%d", index))
                if err != nil {
                        return nil, err
                }
                idVec = append(idVec, blsID)
                blsSig := bls.Sign{}

                err = blsSig.Deserialize(signature)
                if err != nil {
                        return nil, err
                }
                sigVec = append(sigVec, blsSig)
        }
        err := reconstructedSig.Recover(sigVec, idVec) //SlowMist//
        return &reconstructedSig, err
}
```

The `RecoverMasterSig` function utilizes a map (`sigDepositShares`) to store BLS signature shares and then iterates over the map to reconstruct the master signature. However, using a map introduces non-deterministic order during iteration, as the Go programming language does not guarantee a specific order when ranging over map entries. The non-deterministic order can lead to incorrect BLS signature recovery.

Check the logic in `Recover` function, it only take the first item for signature recovery.

```
ret := C.blsSignatureRecover(&sig.v, &sigVec[0].v, (*C.blsId)(&idVec[0].v),
(C.mclSize)(len(idVec)))
```

So do `RecoverValidatorPublicKey/ReconstructSignatures` functions

Solution

Consider using an ordered data structure, such as a slice, to store BLS signature shares to ensure a deterministic order during the signature recovery process. This ensures that the order of shares remains consistent across executions, leading to reliable and correct signature recovery.

Status

Fixed

[N10] [Low] Potential looping issue in `operatorID`

Category: Design Logic Audit

Content

- pkgs/initiator/initiator.go

```
func VerifySharesData(ops map[uint64]Operator, keys []*rsa.PrivateKey, ks *KeyShares,
owner common.Address, nonce uint16) error {
//...
                // Find operator ID by PubKey
                var operatorID uint64
                for id, op := range ops {
                        if bytes.Equal(priv.PublicKey.N.Bytes(), op.PubKey.N.Bytes())
{
                                operatorID = id //SlowMist//
                        }
                }
                sig := secret.SignByte(msg)
                sigs2[operatorID] = sig.Serialize()
//...
}
```

The code snippet in question iterates over a collection of `ops` to find an operator with a matching public key. However, the loop does not include a `break` statement to exit early after finding the first matching operator. This absence of a `break` statement may lead to inefficient execution, as the loop continues to iterate over the remaining elements even after a matching operator has been identified.

Solution

Consider adding a `break` statement immediately after assigning the `operatorID` to exit the loop early once a matching operator is found. This enhances the efficiency of the code by avoiding unnecessary iterations.

Status

Fixed

[N11] [Medium] `x509.IsEncryptedPEMBlock/x509.DecryptPEMBlock` are deprecated

Category: Others

Content

- pkgs/crypto/crypto.go

```
func PrivateKey(path string) (*rsa.PrivateKey, error) {
        data, err := os.ReadFile(path)
        if err != nil {
                return nil, fmt.Errorf("failed to read file: %w", err)
        }

        operatorKeyByte, err := base64.StdEncoding.DecodeString(string(data))
        if err != nil {
                return nil, err
        }
        block, _ := pem.Decode(operatorKeyByte)
        // TODO: resolve deprecation https://github.com/golang/go/issues/8860
        enc := x509.IsEncryptedPEMBlock(block) //nolint //SlowMist//
        b := block.Bytes
        if enc {
                var err error
                // TODO: resolve deprecation https://github.com/golang/go/issues/8860
                b, err = x509.DecryptPEMBlock(block, nil) //nolint //SlowMist//
                if err != nil {
                        return nil, err
                }
        }
        parsedSk, err := x509.ParsePKCS1PrivateKey(b)
        if err != nil {
                return nil, err
        }
        return parsedSk, nil
}
```

Legacy PEM encryption as specified in RFC 1423 is insecure by design. Since it does not authenticate the ciphertext, it is vulnerable to padding oracle attacks that can let an attacker recover the plaintext.

Solution

Using another library

Status

Fixed

[N12] [Information] Logging of sensitive information

Category: Others

Content

- pkgs/dkg/drand.go

```
//#L382
o.Logger.Debug("Encrypted share", zap.String("share", fmt.Sprintf("%x", ciphertext)))
//#L468
o.Logger.Debug("Encrypted share", zap.String("share", fmt.Sprintf("%x", ciphertext)))
```

The code snippet logs sensitive information, specifically the ciphertext, using the `o.Logger.Debug` statement.

Logging such cryptographic material in plaintext may expose sensitive details to logs, which can be a security risk.

Solution

Refrain from logging sensitive information, especially cryptographic material like ciphertext.

Status

Fixed

[N13] [Medium] Delayed handling of errors

Category: Error Unhandle Audit

Content

- pkgs/dkg/drand.go

```
//#L387
depositRootSig, signRoot, err := crypto.SignDepositData(secretKeyBLS,
o.data.init.WithdrawalCredentials, validatorPubKey,
utils.GetNetworkByFork(o.data.init.Fork), MaxEffectiveBalanceInGwei)
o.Logger.Debug("Root", zap.String("", fmt.Sprintf("%x", signRoot)))
// Validate partial signature
val := depositRootSig.VerifyByte(secretKeyBLS.GetPublicKey(), signRoot)
if !val {
    o.broadcastError(err)
    return fmt.Errorf("partial deposit root signature is not valid %x",
depositRootSig.Serialize())
}
```

Performs cryptographic operations and generates a deposit root signature (`depositRootSig`). In case of an error during the cryptographic operations, the error (`err`) is captured and logged, but the handling of the error is delayed until after attempting to verify the signature. This delay might lead to misleading log entries and delayed response to errors.

Solution

Handle errors as soon as they occur to provide accurate and timely information about the operation's success or failure.

Status

Fixed

[N14] [High] Improper use of panic in error handling

Category: Error Unhandle Audit

Content

- pkgs/wire/justification.go

```
func encodeJustifications(justifications []dkg.Justification) ([]encodedJustification,
error) {
        ret := make([]encodedJustification, 0)
        for _, j := range justifications {
                byts, err := j.Share.MarshalBinary()
                if err != nil {
                        panic(err.Error()) //SlowMist//
                }

                ret = append(ret, encodedJustification{
                        ShareIndex: j.ShareIndex,
                        Share:      hex.EncodeToString(byts),
                })
        }
        return ret, nil
}
```

- pkgs/wire/deals.go

```
func EncodeDealBundle(bundle *dkg.DealBundle) ([]byte, error) {
        var publics []string
        for _, p := range bundle.Public {
                byts, err := p.MarshalBinary()
                if err != nil {
                        panic(err.Error()) //SlowMist//
                }
                publics = append(publics, hex.EncodeToString(byts))
        }
        obj := &encodedDeals{
                Deal: &dkg.DealBundle{
                        DealerIndex: bundle.DealerIndex,
```

```
                        Deals:       bundle.Deals,
                        SessionID:   bundle.SessionID,
                        Signature:   bundle.Signature,
                },
                Public: publics,
        }

        return json.MarshalIndent(obj, "", " ")
}
```

- pkgs/operator/operator.go

```
func New(key *rsa.PrivateKey, logger *zap.Logger, dbOptions basedb.Options) *Server {
        r := chi.NewRouter()
        db, err := setupDB(logger, dbOptions)
        // todo: handle error
        if err != nil {
                panic(err) //SlowMist//
        }
        swtch := NewSwitch(key, logger, db)
        s := &Server{
                Logger: logger,
                Router: r,
                State:  swtch,
                DB:     db,
        }
        RegisterRoutes(s)
        return s
}
```

The use of `panic` is not recommended for error handling in production code as it leads to program termination, which can be disruptive and result in loss of data or service interruption.

Solution

Replace the use of `panic` with appropriate error handling mechanisms such as returning errors or logging them for later analysis.

Status

Fixed

[N15] [Medium] Potential slowloris attack due to missing ReadHeaderTimeout configuration

Category: Others

Content

- pkgs/operator/operator.go

```
func (s *Server) Start(port uint16) error {
        srv := &http.Server{Addr: fmt.Sprintf(":%v", port), Handler: s.Router}
        s.HttpServer = srv
        err := s.HttpServer.ListenAndServe()
        if err != nil {
                return err
        }
        s.Logger.Info("✅ Server is listening for incoming requests",
zap.Uint16("port", port))
        return nil
}
```

Start an HTTP server using the `http.Server` type, but it lacks the configuration of the `ReadHeaderTimeout` property. Absence of `ReadHeaderTimeout` may expose the server to potential Slowloris attacks, where an attacker can establish many connections and send headers very slowly, tying up resources and potentially exhausting them.

Solution

Configure the `ReadHeaderTimeout` property for the `http.Server` to mitigate the risk of Slowloris attacks and resource exhaustion.

Status

Fixed

[N16] [Low] Lack of system time accuracy check on service startup

Category: Others

Content

- cli/operator/operator.go
- cli/initiator/initiator.go

The operator service does not perform a check for the accuracy of the system time upon startup. Some service relies on time-sensitive operations (such as generating timestamps, calculate the time spent, etc.), inaccurate system time may lead to operations not executing as expected.

Solution

Perform a check for the accuracy of the system time upon service startup. This can be achieved by obtaining network time, using the Network Time Protocol (NTP) service, or employing other time synchronization mechanisms.

Status

Acknowledged

[N17] [Low] `drand_bls` library is deprecated

Category: Others

Content

- pkgs/wire/dkg.go

```
import (
        ...
        drand_bls "github.com/drand/kyber/sign/bls"
        ...
)
```

"github.com/drand/kyber/sign/bls" is deprecated: This version is vulnerable to rogue public-key attack and the new version of the protocol should be used to make sure a signature aggregate cannot be verified by a forged key. You can find the protocol in kyber/sign/bdn. Note that only the aggregation is broken against the attack and a later version will merge bls and asmbls

Solution

Use another version of the protocol.

Status

Fixed

[N18] [Medium] Lack of Handling Unresponsive Hosts

Category: Design Logic Audit

Content

The `Ping` method in the provided code asynchronously sends ping requests to multiple hosts specified in the `ips` parameter and waits for their responses. However, it only processes and logs responses received from hosts, and

does not handle cases where hosts do not respond. This may lead to unresponsive hosts being overlooked, potentially causing undetected issues or failures.

- pkgs/initiator/initiator.go

```
func (c *Initiator) Ping(ips []string) error {
   client := req.C()
   // Set timeout for operator responses
   client.SetTimeout(30 * time.Second)
   resc := make(chan wire.PongResult, len(ips))
   for _, ip := range ips {
       go func(ip string) {
           resdata, err := c.GetAndCollect(wire.OperatorCLI{Addr: ip},
consts.API_HEALTH_CHECK_URL)
           resc <- wire.PongResult{
               IP:     ip,
               Err:    err,
               Result: resdata,
           }
       }(ip)
   }
   for i := 0; i < len(ips); i++ {
       res := <-resc
       err := c.processPongMessage(res)
       if err != nil {
           c.Logger.Error("😢 Operator not healthy: ", zap.Error(err),
zap.String("IP", res.IP))
           continue
       }
   }
   return nil
}
```

Solution

Ensure that the `Ping` method appropriately handles cases where hosts do not respond within the specified timeout period. This can be achieved by implementing a mechanism to detect and log unresponsive hosts or by setting a reasonable timeout for each request and handling timeout errors explicitly.

Status

Fixed

[N19] [Medium] Lack of Handling All Switch Cases and Missing Default Case

Category: Design Logic Audit

Content

The `VerifyIncomingMessage` method in the provided code uses a switch statement to handle different message types (`PingMessageType` and `ResultMessageType`). However, it lacks a `default` case to handle unexpected message types, and does not cover all possible cases that may occur in the switch statement. This can lead to unexpected behavior or errors if messages of unrecognized types are received.

- pkgs/operator/state.go

```
func (s *Switch) VerifyIncomingMessage(incMsg *wire.SignedTransport) (uint64, error) {
   var initiatorPubKey *rsa.PublicKey
   var ops []*wire.Operator
   var err error
   switch incMsg.Message.Type {
   case wire.PingMessageType:
       ping := &wire.Ping{}
       if err := ping.UnmarshalSSZ(incMsg.Message.Data); err != nil {
           return 0, err
       }
       // Check that incoming message signature is valid
       initiatorPubKey, err = crypto.ParseRSAPublicKey(ping.InitiatorPublicKey)
       if err != nil {
           return 0, err
       }
       ops = ping.Operators
       err = s.VerifySig(incMsg, initiatorPubKey)
       if err != nil {
           return 0, err
       }
   case wire.ResultMessageType:
       resData := &wire.ResultData{}
       if err := resData.UnmarshalSSZ(incMsg.Message.Data); err != nil {
           return 0, err
       }
       s.Mtx.RLock()
       inst, ok := s.Instances[resData.Identifier]
       s.Mtx.RUnlock()
       if !ok {
           return 0, utils.ErrMissingInstance
       }
       msgBytes, err := incMsg.Message.MarshalSSZ()
       if err != nil {
           return 0, err
       }
```

```
        // Check that incoming message signature is valid
        err = inst.VerifyInitiatorMessage(msgBytes, incMsg.Signature)
        if err != nil {
            return 0, err
        }
        ops = resData.Operators
    }
    operatorID, err := spec.OperatorIDByPubKey(ops, s.PubKeyBytes)
    if err != nil {
        return 0, err
    }
    return operatorID, nil
}
```

Solution

Implement Default Case.

Status

Fixed

[N20] [Information] Mixed Value and Pointer Receivers

Category: Others

Content

The `MarshalJSON` and `UnmarshalJSON` methods in the provided code use a mix of value and pointer receivers (`OperatorCLI` and `*OperatorCLI`). This inconsistency in receiver types can lead to confusion and potential bugs, as it violates the principle of consistency and may cause unexpected behavior.

- pkgs/wire/types_json.go

```
func (o OperatorCLI) MarshalJSON() ([]byte, error) {
    pk, err := EncodeRSAPublicKey(o.PubKey)
    if err != nil {
        return nil, err
    }
    return json.Marshal(operatorCLIJSON{
        Addr:   o.Addr,
        ID:     o.ID,
        PubKey: string(pk),
    })
}

func (o *OperatorCLI) UnmarshalJSON(data []byte) error {
```

```
    var op operatorCLIJSON
    if err := json.Unmarshal(data, &op); err != nil {
        return fmt.Errorf("failed to unmarshal operator: %s", err.Error())
    }
    _, err := url.ParseRequestURI(op.Addr)
    if err != nil {
        return fmt.Errorf("invalid operator URL %s", err.Error())
    }
    pk, err := ParseRSAPublicKey([]byte(op.PubKey))
    if err != nil {
        return fmt.Errorf("invalid operator public key %s", err.Error())
    }
    *o = OperatorCLI{
        Addr:   strings.TrimRight(op.Addr, "/"),
        ID:     op.ID,
        PubKey: pk,
    }
    return nil
}
```

Solution

To maintain consistency and clarity, it's recommended to use either all value receivers or all pointer receivers for methods operating on a particular struct type. Choose one approach based on the requirements and design of the codebase.

Status

Fixed

[N21] [Low] Lack of HTTPS Usage

Category: Design Logic Audit

Content

The `Start` method in the provided code starts an HTTP server to listen for incoming requests on the specified port (`port`). However, it only sets up an HTTP server without enforcing the use of HTTPS, which can lead to security risks such as data interception, manipulation, or unauthorized access, especially when handling sensitive information or user authentication.

docker-compose.yml

```
    command:
      [
```

```
        "ping",
        "--ip",

"http://operator1:3030,http://operator2:3030,http://operator3:3030,http://operator4:30
30",
      ]
```

Solution

Enabling HTTPS ensures that all communication between clients and the server is encrypted, providing confidentiality and integrity for sensitive data.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002312110001	SlowMist Security Team	2023.11.28 - 2023.12.11	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 1 high risk, 8 medium risk, 6 low risk vulnerabilities. All the findings were fixed.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist