DDAM Baseline Security Audit Report

Contents

1 Executive Summary

SlowMist Security Team received an application from the DDAM team for the DDAM security audit on 2019-10-15. SlowMist Security Team developed an audit plan based on the agreement between the parties and the project characteristics and finally issued a security audit report as follows.

SlowMist Security Team conducts a complete security test of the project in the most close-to-true attack.

SlowMist Blockchain System Test Method:

Black-box Test	Conduct a security test from an attacker's perspective.
Gray-box Test	Security testing of code modules through scripting tools, observing internal operational status and identifying weaknesses.
White-box Test	Based on open source and non-open source code, detect whether there are vulnerabilities in programs such as nodes, SDKs etc.

SlowMist Blockchain Risk Level:

Critical Vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the blockchain, and it is highly recommended to fix serious vulnerabilities.
High-Risk Vulnerabilities	vulnerabilities will affect the normal operation of the blockchain, and it is highly recommended to fix high-risk vulnerabilities.
Medium-Risk Vulnerabilities	Medium-risk vulnerabilities will affect the operation of blockchains and it is recommended to fix medium-risk vulnerabilities.

Low-Risk Vulnerabilities	Low-risk vulnerabilities can affect the operation of blockchains in specific scenarios. It is recommended that the developer team evaluate and consider whether these issues need to be fixed.
Weaknesses	There are security risks in theory, but they are extremely difficult to repeat.
Improve Recommendations	There are better implementations of coding or architecture.

2 Context

2.1 Description

DDAM is a next generation of data economy, transform data resource into data asset. (From official website)

Official Website：https://ddam.one

Source Code：Undisclosed

Initial Audit Version：

SHA256(ddamchain_pack2.zip)=

78d3a34f997fda137148dc40fd010287698e618410933bf6d889c96a00ef79fc

Review Version：

Commit: 3871aa7b8fb6be548e819db8532bcfc064158a3f

2.2 Audit Scope

The main items of this security audit include:

No.	Audit Item	Audit Subclass	Audit Result
1	P2P Security	Audit the Number of Node	Passed

		Connections	
		Node Performance Audit	Passed
		Communication Encryption Audit	Passed
		Alien Attack Audit	Passed
2	RPC Security	RPC Permission Audit	Passed
		Malformed Data Request Audit	Passed
		Communication Encryption Audit	Passed
		Same Origin Policy Audit	Passed
3	Encrypted Signature Security	Random Number Generation Algorithm Audit	Passed
		Private Key Storage Audit	Passed
		Cryptography Component Call Audit	Passed
		Hash Strength Audit	Passed
		Length Extension Attack Audit	Passed
		Encryption/Decryption Fuzzing Test	Passed
4	Account System and Transaction Model Security	Authority Verification Audit	Passed
		Transaction Replay Audit	Passed
		“False top-up”Audit	Passed
5	PoC Consensus Security	Block Verifucation	Passed
		Grinding Block Attack Audit	Passed
		PoW Computing Competition Audit	Passed

(Other unknown security vulnerabilities are not covered by this audit responsibility)

3 Code Overview

3.1 Architecture

It is built on the basis of the PoC consensus. Its main feature is to use hard disk storage space as a consensus participant, and then use the slow hash function to resist the computing power competition of PoW, greatly reduce the dependence of cryptocurrency on energy consumption, and achieve the goal of green environmental protection, low energy consumption, and low noise. The PoC consensus was first applied to Burstcoin.

3.2 P2P Security

3.2.1 Number of Node Connections Audit

If there is no connection number configuration, the node may experience severe performance degradation due to too many connections.

After feedback from the developer team, it is confirmed that the TCP protocol requires the communication parties to establish a connection. If the number of connections is too large, the system performance will be degraded. The communication of DDAM chain data is mainly based on UDP protocol, and UDP protocol is a connectionless protocol (the cost of connection maintenance system is negligible), which is different from TCP protocol which needs to establish a connection and consume system resources. Moreover, the messages in DDAM support priority, so this issue has little impact on the DDAM chain. There is no single IP connection limit, which is easy to cause Sybil attacks.

3.2.2 Node Performance Audit

There is no rate limit for node data synchronization. When there are more nodes to synchronize, the performance of the node may be seriously degraded, and even the synchronization and the outbound block may be affected.

After feedback from the developer team, it was confirmed that the DDAM chain data communication is mainly based on the UDP protocol, and the UDP protocol does not need to establish a connection (more system resource utilization) like the TCP protocol. Moreover, the messages in the DDAM support the priority, so even in the case of heavy communication, messages with high priority such as block and synchronization will be processed preferentially, which will not cause an obstacle to the impact of the chain service.

3.2.3 Communication Encryption Audit

Unencrypted communication brings privacy risks, security risks and integrity risks to network participants。

3.2.4 Alien Attack Audit

Nodes of different chains will judge the chainID during the handshake process, so that they will not be connected to each other and will not cause address pool pollution.

- network/net.go

```
func (nc *NetCore) onHandleDataMessage(data *MsgData, fromID NodeID) {
    if atomic.LoadInt32(&nc.unhandledDataMsg) > MaxUnhandledMessageCount {
        Logger.Info("unhandled message too much , drop this message !")
        return
    }
```

```
    chainID, protocolVersion := decodeMessageInfo(data.MessageInfo)
    p := nc.peerManager.peerByID(fromID)
    if p != nil {
        p.chainID = chainID
        if !p.IsCompatible() {
            Logger.Info("Node chain ID not compatible, drop this message !")
            return
        }

        if !p.isAuthSucceed {
            Logger.Info("Peer Authentication is not succeed , drop this message !")
            return
        }
    }

    if netServerInstance != nil {
        netServerInstance.handleMessage(data.Data, fromID.GetHexString(), chainID, prot
ocolVersion)
    }

}
```

3.3 RPC Security

3.3.1 RPC Permission Audit

The RPC port of the wallet is not open by default.

The RPC port of the node has no asset-related sensitive permission interface.

3.3.2 Malformed Data Request Audit

The test sent a complicated request, an oversized JSON, and the exception packet did not crash.

3.3.3 Communication Encryption Audit

Unencrypted communication brings privacy risks, security risks and integrity risks to network participants。

3.3.4 CORS Policy Audit

The RPC port of the wallet is not open by default.

The RPC port of a node is enabled cross-domain by default, but does not affect asset security.

3.4 Encrypted And Signature Security

3.4.1 Random Number Generation Algorithm Audit

The generation of the private key seed is based on the crypto/rand standard library, and the safety of entropy value.

- crypto/privatekey.go

```
func GenerateKey(s string) (PrivateKey, error) {
    var r io.Reader
    if len(s) > 0 {
        r = strings.NewReader(s)
    } else {
        r = rand.Reader
    }
    var pk PrivateKey
    _pk, err := ecdsa.GenerateKey(getDefaultCurve(), r)
    if err == nil {
        pk.PrivKey = *_pk
    } else {
        err = fmt.Errorf("GenKey Failed, reason : %v.\n", err.Error())
        return pk, err
    }
    return pk, nil
}
```

3.4.2 Keystore Audit

The wallet does not perform password strength detection, allowing the use of weak passwords to store private keys.

The program running on the node is not separated from the mining program. The private key needs to be stored on all mining machines and is not safe to manage.

3.4.3 Cryptographic Component Call Audit

No erroneous calls were found.

3.4.4 Hash Strength Audit

It has not been found that weak hash functions such as md5 and sha1 are used for encryption.

3.4.5 Length Extension Attack Audit

The length extension vulnerability of the hash function could not be found.

3.4.6 Crypto Fuzzing Test

The hard disk capacity proof algorithm is based on the Shabal256 algorithm, which tests:

```
func TestSum2556(t *testing.T) {
    for index := 0; index < 100000; index++ {
        for _, test := range shabal256Tests {
            s := fmt.Sprintf("%X", Sum256([]byte(test.in)))
            if s != test.out {
                t.Fatalf("Sum256 function: shabal256(%s) = %s want %s", test.in, s, tes
t.out)
            }
        }
    }
}
```

On a Macbook Pro A1706, the single-threaded test shabal256 efficiency, the data shows about 7.7w times/sec, equivalent to 18 Nonce/sec (1 nonce generated contains 4096 Scoop, requires

4097 shabal256 operations). Assume that each Scoop occupies 256k of storage, which is approximately equal to 4.7M of storage. From the mining difficulty of other similar PoC public chains on the market, it is necessary to have a PB-level computing power to have a large probability of generating blocks. It is practically impossible to generate the calculation amount of the PB-level hard disk space in a limited time.

3.5 Account and Transaction Model Security

3.5.1 Authority Verification Audit

Types and values are strictly checked for each field such as signature, type, and Nonce in the transaction structure.

- core/tx_validate.go

```
func getValidator(tx *types.Transaction, validateState bool) validator {
    return func() error {
        var err error
        // Common validations
        if err = commonValidate(tx); err != nil {
            return err
        }

        // Validate gas
        if err = gasValidate(tx); err != nil {
            return err
        }

        // Recover source at last for performance concern
        if err := sourceRecover(tx); err != nil {
            return err
        }

        // Validate state
        if validateState {
            db := BlockChainImpl.LatestStateDB()
            if err = stateValidate(db, tx, BlockChainImpl.Height()); err != nil {
                return err
```

```
            }
        }

        switch tx.Type {
        case types.TransactionTypeTransfer, types.TransactionTypeStakeAdd, types.Transa
ctionTypeStakeReduce:
            err = valueValidate(tx)
        case types.TransactionTypeBindUMID, types.TransactionTypeTransformUMID, types.T
ransactionTypeUnbindUMID:
            if tx.Source.String() != administratorAddress.String() {
                return fmt.Errorf("Not an administrator, no permission !")
            }
            err = dataValidate(tx)
        }
        if err != nil {
            return err
        }

        return nil
    }
}
```

3.5.2 Replay Attack Audit

The field participating in the signature contains Nonce, and transactions on the same chain cannot be replayed;

The field participating in the signature does not contain the chainID, and the same chain or the main network and the test network may cause a transaction replay attack.

- global/types/tx_hashing.go

```
func (th *txHashing) genHash() common.Hash {
    buf := &bufferWriter{}
    buf.writeBytes(th.target)
    buf.writeBytes(th.value)
    buf.writeBytes(th.nonce)
    buf.writeBytes(th.gasLimit)
    buf.writeBytes(th.gasPrice)
```

```
    buf.writeByte(th.typ)
    buf.writeBytes(th.data)
    return common.BytesToHash(common.Sha256(buf.Bytes()))
}
```

3.5.3 "False Top-up" Audit

There is no mechanism like EOS Hard Failed.

There is no mechanism like EVM return fail.

There is no mechanism like USDT OP_RETURN.

There are no fields like Ripple deliver_amount.

3.6 PoC Consensus Security

3.6.1 Block Verification Audit

Block verification does not limit the minimum block time, and the malicious miner can construct time to cause the block time to advance.

- consensus/engine.go

```
func (e *Engine) VerifyBlockHeader(bh *types.BlockHeader) (bool, error) {
    if e == nil {
        return false, fmt.Errorf("engine not initialize")
    }
    if bh.Hash != bh.GenHash() {
        return false, fmt.Errorf("hash diff")
    }
    // Checks auth
    ok, err := e.auth.VerifyAuthCode(bh.Auth, bh.Proposer, bh.PreHash)
    if err != nil {
        return false, fmt.Errorf("verify auth code error:%v", err)
    }
    if !ok {
        return false, fmt.Errorf("the address cann't propose:%v", bh.Proposer)
    }
    // Checks if the block time is before current time
    if bh.CurTime.After(global.Context().TimeService.Now()) {
```

```
        return false, fmt.Errorf("block too early, now %v, block time %v", global.Conte
xt().TimeService.Now().Local(), bh.CurTime.Local())
    }

    // Verifies the signature
    pk, err := bh.Sign.RecoverPubkey(bh.Hash.Bytes())
    if err != nil {
        return false, fmt.Errorf("recover pk fail:%v", err)
    }
    addr := pk.GetAddress()
    if !addr.EqualTo(bh.Proposer) {
        return false, fmt.Errorf("address diff, expect %v, infact %v", addr, bh.Propose
r)
    }

    return true, nil
}
```

3.6.2 Grinding Block Attack Audit

When the miners produce blocks, they can change the signature value Sign by constructing the block content, greatly improving their probability of winning in the next block.

```
// VerifyBlockHeaderPair verifies if the block is legal on behalf consensus
func (e *Engine) VerifyBlockHeaderPair(prevBH, bh *types.BlockHeader) (bool, error) {
    if ok, err := e.VerifyBlockHeader(bh); !ok {
        return false, err
    }
    if bh.Height != prevBH.Height+1 {
        return false, fmt.Errorf("height error:%v %v", prevBH.Height, bh.Height)
    }
    scoopNumber := scoopNum(prevBH)
    // Generates the specified scoop
    scoop := e.plotter.GenerateScoop(bh.Proposer, bh.Nonce, scoopNumber)
    if scoop == nil {
        return false, fmt.Errorf("generate scoop error %v-%v-%v", bh.Proposer, bh.Nonce,
 scoopNumber)
    }

    // Checks the deadline
    dl := deadline(prevBH, scoop)
```

```
    if !deadlineLegal(prevBH.CurTime, global.Context().TimeService.Now(), dl) {
        return false, fmt.Errorf("deadline not ok: %v %v", dl, global.Context().TimeSer
vice.Since(prevBH.CurTime))
    }
    // Checks the target and difficulty
    target, diff := e.calcBaseTarget(prevBH, bh.CurTime)
    if target != bh.BaseTarget {
        return false, fmt.Errorf("target error:%v %v", target, bh.BaseTarget)
    }
    if diff.Cmp(bh.CumulativeDifficulty) != 0 {
        return false, fmt.Errorf("difficulty error : %v %v", diff, bh.CumulativeDifficu
lty)
    }
    return true, nil
}
```

3.6.3 PoW Computing Competition Audit

The PoC Scoop generation algorithm generates the PoC1 format. Historically, this caused Burstcoin to be attacked. It is recommended to upgrade to PoC2.

- consensus/plotter/generator.go

```
func GenerateScoop(address []byte, nonce uint64, scoopNum int) *Scoop {

    if scoopNum >= ScoopNumPerNonce {
        return nil
    }
    var gendata [16 + NonceSize]byte

    if len(address) != 32 {
        return nil
    }
    seedID := genSeedID(address)
    var seedIDBytes = make([]byte, 8)
    binary.LittleEndian.PutUint64(seedIDBytes, uint64(seedID))

    for i := 0; i < 8; i++ {
        gendata[NonceSize+i] = seedIDBytes[7-i]
    }
    var nonceBytes = make([]byte, 8)
```

```
    binary.LittleEndian.PutUint64(nonceBytes, uint64(nonce))

    for i := 8; i < 16; i++ {
        gendata[NonceSize+i] = nonceBytes[15-i]
    }

    for i := NonceSize; i > 0; i -= HashSize {

        len := NonceSize + 16 - i
        if len > HashCap {
            len = HashCap
        }

        actualBytes := common.Shabal256(gendata[i : i+len])
        copy(gendata[i-HashSize:i], actualBytes)
    }

    final := common.Shabal256(gendata[:])

    // XOR with final
    for i := 0; i < NonceSize; i++ {
        gendata[i] ^= final[i%HashSize]
    }
    scoop := &Scoop{}
    hashStart := scoopNum * HashSize * 2
    scoop.Data[0] = common.BytesToHash(gendata[hashStart : hashStart+HashSize])
    hashStart += HashSize
    scoop.Data[1] = common.BytesToHash(gendata[hashStart : hashStart+HashSize])
    return scoop
}
```

Reference: https://burstwiki.org/en/technical-information-to-create-plot-files/

After feedback from the developer team and reference to the technical white paper, we confirmed that the data generation method similar to POC2 is used: this requires calculating the shabal value of all 8192 hash groups corresponding to the nonce as the final hash, and doing the Xor operation on all the hash groups to get the final Hash value. This will not cause problems that same with PoC1.

4 Result

4.1 Critical Vulnerabilities

- When the miners produce blocks, they can change the signature value Sign by constructing the block content, greatly improving their probability of winning in the next block.
  (This issue has been fixed in the review version)
- Block verification does not limit the minimum block time, and the malicious miner can construct time to cause the block time to advance.
  (This issue has been fixed in the review version)

4.2 High-Risk Vulnerabilities

- The field participating in the signature does not contain the chainID, and the same chain or the main network and the test network may cause a transaction replay attack.
  (This issue has been fixed in the review version)

4.3 Medium-Risk Vulnerabilities

- There is no single IP connection limit, which is easy to cause Sybil attacks.
  (This issue has been fixed in the review version)

4.4 Improve Recommendations

- It is recommended that the wallet be tested for password strength to avoid using weak passwords.

- The private key needs to be stored on all mining machines, which is extremely unsafe in management. It is recommended to support the mining program and node program separation.
- P2P unencrypted communications introduce privacy, security and integrity risks to network participants and recommend improvements.
- RPC unencrypted communications introduce privacy, security and integrity risks to network participants and recommend improvements.

4.5 Exchange Security Summary

- When the user recharges the account, make sure that the block has enough acknowledgments and the transaction status is "success" and the transaction type: type=0. Pay attention to the accuracy of the token in this process.

4.7 Conclusion

Audit Result: Passed

Audit No: BCA001910240001

Audit Date: October 24, 2019

Audit Team: SlowMist Security Team

Conclusion: All problems found have been fixed after feedback and developer team revisions. DDAM Chain does not have the risks mentioned above.

5 Disclaimer

This audit report does not imply any warranty on the security of the project. SlowMist issues this report with reference to the facts of the project that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility. SlowMist is not responsible for unknown vulnerabilities and security incidents that happens after the report is issued, because there is no guarantee that the security status of the system after the report is issued. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). If the information provided is missing, tampered, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. This report is available in Chinese and English. When an ambiguity occurs, the interpretation of the Chinese version shall prevail.

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号