Accumulate Network

AUDIT REPORT

Version 1.0.0

Serial No. 20220916000012025

Presented by Fairyproof

September 16, 2022

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Accumulate Network project.

Audit Start Time:

July 7, 2022

Audit End Time:

September 13, 2022

Audited Code's GitLab Repository:

https://gitlab.com/accumulatenetwork/accumulate

Audited Code's GitLub Commit Number When Audit Started:

44fd61266989cb736050ba35db3e73ec23d220f1

Audited Code's GitLub Commit Number When Audit Ended:

3df599108ad796c070b57d4696577c88fe248dea

Goal:

The goal of this audit is to review Accumulate Network's Golang implementation for a blockchain , study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

Scope:

The files under the "cmd", "config", "internal", "networks", "pkg/client/signing", "protocol" and "smt" at https://gitlab.com/accumulatenetwork/accumulate/ are to be audited.

Comprehensive Review or Test of Protocol Architecture Implementation

Cryptography:

- Cryptographic Libraries used

- PBT/Merkle Tree

- Signing Verification

- Signature Chain

Key Management:

- Key Book Account

- Key Page Account

- Key Generation

- Key Storage

Identities (Accumulate Digital Identifier:

- ADI

- Sub-ADI

- Data Account

- Token Account

- Scratch Accounts

- Token Issuance

Chain Implementation/Consensus Review:

- Chain of Chains Review (BVNs to DN) Implementation of Tendermint

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Accumulate Network team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the Accumulate Network should work:

White-paper: https://accumulatenetwork.io/whitepaper

Development Docs: https://docs.accumulatenetwork.io/

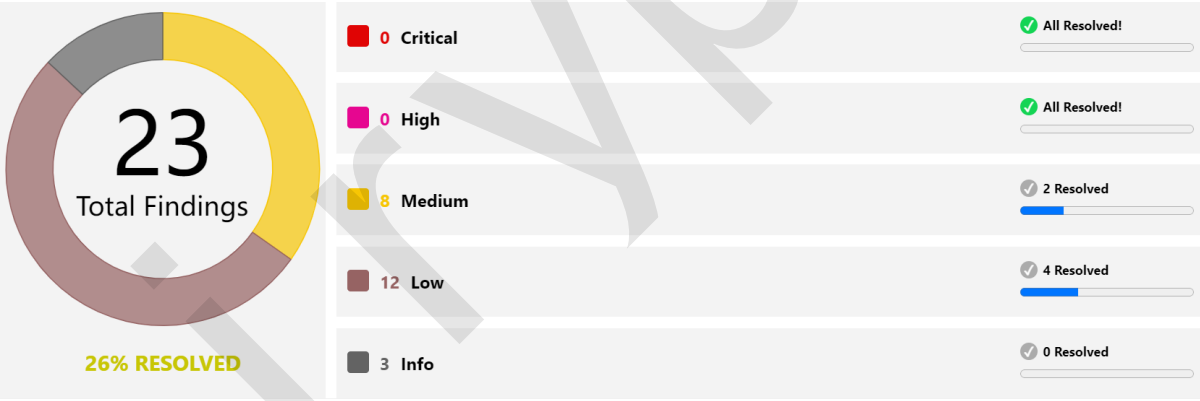
Tendermint - Consensus without Mining: https://tendermint.com/static/docs/tendermint.pdf

Tendermint: https://docs.tendermint.com/

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Accumulate Network team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022091600012025	Fairyproof Security Team	July 7, 2022 - Sep 13, 2022	Low Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, 8 issues of medium-severity, 12 issues of low-severity and 3 issues of informational-severity were uncovered. The Accumulate team fixed 2 issues of medium-severity and 4 issues of low-severity, and acknowledged the remaining issues.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Accumulate Network

Accumulate Network is an identity-based blockchain protocol with multi-chain support, human-readable addresses, and key hierarchies. It has the following features:

An Identity-Based Blockchain Protocol: Accumulate is a new kind of blockchain protocol that is organized completely around identities. Accumulate Digital Identifiers (ADIs) can be assigned to organizations, devices people, or things.

Multi-Chain Support: Accumulate's multi-chain architecture enables greater throughput. The interconnected network of chains results in 70,000 transactions per second (TPS) which makes it one of the fastest protocols.

Human-Readable Addresses: A human-readable address enables users to transact with a static address such as acc://Bob.acme. This is in contrast to traditional random generated addresses such as 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa.

Key Hierarchies: Key hierarchies enable enterprise-grade security. Accumulate users can rotate, recreate, and reassign private keys when necessary. Using key hierarchies results in greater safety of assets.

Accumulate uses the Tendermint API and config files to set up networks. The Tendermint consensus will be run for the Block Validator Network (BVN) and Directory Network (DN) in the Accumulate network so that adding BVNs adds capacity linearly up to 1,000 TPS per BVN.

04. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Memory Leak
- Unreleased Resources
- Overflow/Underflow
- Concurrency
- Integer Truncation
- Inappropriate Setting
- Go-routine Unsafe Exiting
- Closing Channel Twice
- Missing Check for Unsafe External Input
- Unsafe Encryption
- Unsafe Random Number Generation
- Data Security
- DOS Attack
- Security in Consensus Algorithm
- Handling of Block
- Transaction Security
- Database Security
- Oracle Security
- Security of Third-party Libraries
- Logic Vulnerability
- Code Improvement
- Misc

05. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

06. Major areas that need attention

Based on the provided files the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Incentive & Slashing Mechanisms

We checked whether there were issues or risks in the incentive and slashing mechanism.

According to the white-paper describes, a DPOS algorithm should be implemented. However in the existing implementation, this algorithm hasn't been implemented yet. Therefore no incentive or slashing mechanism is implemented.

For more details, please refer to FP-1, FP-2 and FP-15 in "08. Issue descriptions"

- Implementation of ABCI

We checked if there were issues with the implementation of ABCI.

We found some issues, for more details please refer to FP-3, FP-4 and FP-7 in "08. Issue descriptions".

- Execution of Transactions

We checked if there were issues with execution of transactions.

Since Accumulate is a multi-chain network, a transaction may be executed across multiple sub-chains. If a transaction fails to be executed it will be re-executed. However this mechanism cannot guarantee atomic executions.

In addition, if the validators in the sub-chains act maliciously, the whole network will suffer from huge risks.

We found some issue, for more details please refer to FP-13, FP-5, FP-6, FP-10 and FP-11 in "08. Issue descriptions".

- Account & Signature Schemes

We check whether there were issues with Accumulate's account and signature schemes.

Accumulate uses ADI ad IDs. When an ADI is created, it should be validated.

The existing implementation of signature scheme could be refined and restructured.

The existing implementation only supports the ED25519 algorithm.

We found some issues, for more details please refer to FP-16, FP-18, FP-19 and FP-8 in "08. Issue descriptions".

- Code Quality

We checked whether there were issues with the implementation's code quality.

The implementation has been tested with unit tests, simulation tests and fuzz tests.

However we found one issue, for more details please refer to FP-22 in "08. Issue descriptions".

- Technical Documentation

It's better to update the white paper and add the latest updates in the technical documents. This helps both developers and users to understand the project better.

For more details please refer to FP-21 in "08. Issue descriptions".

- Code Comments

Accumulate is a multi-chain network, its implementation such as execution of transactions, account and signature schemes are complex. To help developers and auditors better understand it, it would be better to add more comments.

For more details please refer to FP-23 in "08. Issue descriptions".

- Miscellaneous

More functions may be needed for CLI and walletd.

The implementation of HTTP RPC API doesn't handle time-out.

The Oracle's security needs to be reinforced.

Panic may cause DOS attacks.

For more details about these please refer to FP-9, FP-12, FP-14 and FP-20 in "08. Issue descriptions".

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	DPOS Algorithm Not Implemented	Implementation Vulnerability	Medium	Acknowledged
FP-2	No Slashing for Tendermint Validators	Design Vulnerability	Medium	Acknowledged
FP-3	Missing Consistency Check for Validators	Security in Consensus Algorithm	Medium	✓ Fixed
FP-4	Unnecessary checkTxMutex in CheckTx	Concurrency	Medium	Acknowledged
FP-5	Potential Malicious Leader	Security in Consensus Algorithm	Medium	✓ Fixed
FP-6	Potential Malicious BVN	Security in Consensus Algorithm	Medium	Acknowledged
FP-7	Possible Replay Attacks	Security in Consensus Algorithm	Medium	Acknowledged
FP-8	Inappropriate Setting in Timestamp	Inappropriate Setting	Medium	Acknowledged
FP-9	Incomplete CLI Functions	Implementation Vulnerability	Low	Acknowledged
FP-10	Missing Validation for Parameters	Data Security	Low	✓ Fixed
FP-11	Missing Validation for Parameters	Data Security	Low	✓ Fixed
FP-12	Inappropriate Configurations	Data Security	Low	✓ Fixed
FP-13	Issue with Atomic Transactions	Transaction Security	Low	Acknowledged
FP-14	Oracle Security	Oracle Security	Low	Acknowledged
FP-15	Inappropriate Validator Power	Security in Consensus Algorithm	Low	Acknowledged
FP-16	Inappropriate Characters	Misc	Low	Acknowledged

Index	Title	Issue/Risk	Severity	Status
FP-17	Signature Schemes Could be Improved	Code Improvement	Low	Acknowledged
FP-18	Reinforcing Token Security	Code Improvement	Low	Acknowledged
FP-19	Limitation in Signature Algorithms	Code Improvement	Low	Acknowledged
FP-20	Inappropriate Error Handling	Code Improvement	Low	✓ Fixed
FP-21	White-paper Not Updated	Misc	Info	Acknowledged
FP-22	More Static Testing Needed	Misc	Info	Acknowledged
FP-23	More Comments Needed	Misc	Info	Acknowledged

08. Issue descriptions

[FP-1] DPOS Algorithm Not Implemented Medium Acknowledged

Issue/Risk: Implementation Vulnerability

Description:

At the time of audit, Accumulate's DPOS algorithm hasn't been implemented.

Recommendation:

Consider implementing a DPOS algorithm including staking, binding/unbinding, slashing based on Tendermint Core

Update:

The Accumulate team plans to implement it in its layer 2 solution.

Status:

It has been acknowledged by the Accumulate team.

[FP-2] No Slashing for Tendermint Validators

Medium Acknowledged

Issue/Risk: Design Vulnerability

Description:

When the Tendermint consensus discovers an evidence, no slashing is applied to the validators

Recommendation:

Consider implementing a slashing mechanism to punish malicious validators.

Update:

An evidence is recorded on a data chain. A layer 2 staking system will be responsible for discovering an evidence and punishing the misbehaving validators.

Status:

The Accumulate team has acknowledged this and plans to implement it in future upgrades.

[FP-3] Missing Consistency Check for Validators

Medium ✓ Fixed

Issue/Risk: Security in Consensus Algorithm

Description:

initChain didn't check the consistency between the Tendermint's validators and Genesis' validators.

Recommendation:

Consider comparing Tendermint's returned validators with genesis' validators, if they are inconsistent, the blockchain can't be started.

And consider returning the new validators' information which is read from globals to Tendermint.

Update:

It has been fixed at https://gitlab.com/accumulatenetwork/accumulate/-/commit/77d1e5d79d799ea6e310cc5b9fea4ae59be5c5e8 .

Status:

It has been fixed by the Accumulate team.

[FP-4] Unnecessary checkTxMutex in CheckTx

Medium Acknowledged

Issue/Risk: Concurrency

Description:

`CheckTX` used `checkTxMutex` to handle concurrency in `checkTxBatch`. This was unnecessary since Tendermint has a global mutex to handle concurrency.

Recommendation:

Consider removing `checkTx`

Update/Status:

The Accumulate team has acknowledged this issue

[FP-5] Potential Malicious Leader Medium ✓ Fixed

Issue/Risk: Security in Consensus Algorithm

Description:

When a synthetic transaction was sent from a leader to a BVN, this leader sent an anchor to the DN. When the BVN received the transaction it would firstly verify the anchor and then execute the transaction. If the leader was a malicious actor, it could send a fake transaction and a fake anchor resulting in the BVN to verify the fake anchor and then execute the fake transaction.

Recommendation:

Consider sending synthetic transactions from leaders but sending anchors from other validators, and only after an anchor has been verified by 2/3 of the validators, can this anchor be considered as a valid anchor.

Update:

Accumulate implemented a 2/3 solution, but needed to do testing to ensure it will not cause performance issues.

Status:

It has been fixed by the Accumulate team in https://gitlab.com/accumulatenetwork/accumulate/-/commit/c5a68005cf44942f3dcf7b7b97bdddf96d3abc6d

[FP-6] Potential Malicious BVN Medium Acknowledged

Issue/Risk: Security in Consensus Algorithm

Description:

When 2/3 of the validators within a single BVN were malicious, this BVN would be considered as a malicious BVN. It is possible the malicious BVNs could attack normal BVNs. Especially when the total number of validators within a BVN were few, this BVN would be easily manipulated and used to attack the whole system.

Recommendation:

Consider ensuring that each single BVN has at least a certain number of validators. Consider using a well designed algorithm to randomly allocate validators for each single BVN.

Update:

At launch, each BVN will have at least 10 validators, and those will be spread across the different mainnet participants. the Accumulate team intends to implement something more robust (including randomization) for 1.1.

Status:

The Accumulate team has acknowledged this issue

[FP-7] Possible Replay Attacks Medium Acknowledged

Issue/Risk: Security in Consensus Algorithm

Description:

After a user submits a transaction to a Tendermint node, the node will call CheckTx to verify the transaction by:

a) Checking whether the user has sufficient Credits to pay for the transaction fee (If the user doesn't have sufficient Credits, this transaction will fail.) and

b) Executing the transaction

In real cases, a block quite often contains multiple transactions which may rely on one another, so the verification of one transaction may actually need to verify multiple transactions. But CheckTx doesn't handle this case and can only verify a single transaction. Furthermore, CheckTx may not be able to properly check Credits in this case. These may result in the verification of a transaction to fail. A transaction that doesn't go through CheckTx's verification will not be broadcast to the Tendermint's mempool and may be used for replay attacks. A malicious actor can collect these failed transactions, wait until a transaction goes through CheckTx's verification and rebroadcast these failed transactions to launch replay attacks.

Recommendation:

Consider implementing Tendermint's gas mechanism, don't execute a transaction in CheckTx, and only check the transaction's format and signature in CheckTx. However this may give chances to malicious actors to launch DDOS attacks.

Update:

The Accumulate notes that gas fees are a mechanism to specify a maximum fee because contracts can't be evaluated for how much CPU they require without running them. Then gas refunds what isn't used.

We don't do contracts, so the execution profile of transactions is fixed. So a function of the transaction and bytes in the transaction is all we need. Our credit mechanism is nice but not that different than fees in bitcoin or really gas for strictly native token transactions.

Currently invalid token transactions are rejected by any of the networks (BVN, DN).

Same if the credits to pay don't exist.

Once the txs pass CheckTX , the only failure possible in DeliverTX is that multiple tx drain the credits. But in the next block (1 second later) all transactions will fail unless credits are added.

An actual business use case will not publish a failed transaction to the entire network, but just the node they submit the transaction to, and (if relayed) the node on the network that gets the tx first. So the risk of a replay only applies to:

1. A transaction with insufficient credits (But a wallet can check this prior to submission)
2. Multiple submission of transactions that drain the credits (more like an attack)
3. The node (s) that the transaction is submitted to are the only nodes that see a failed tx
4. A transaction that the user is not interested in resubmitting if the tx fails
5. A transaction that is not followed by a valid transaction that increments the signature nonce

All in all, this doesn't look to me to be a significant security risk. If an Exchange is submitting a tx that fails do to lack of credits, the current system allows the credits to be repaired and the transaction to be resubmitted. This appears to me to be the most likely scenario.

Status:

The Accumulate team has acknowledged this issue.

[FP-8] Inappropriate Setting in Timestamp Medium Acknowledged

Issue/Risk: Inappropriate Setting

Description:

Accumulate uses timestamps to prevent replay attacks. When an account submits a transaction, a timestamp will be stamped on the signature.

LastUsedOn can be set to MaxUint64, if the signing account is Lite account, it will cause all new signatures to fail.

Recommendation:

Consider setting the signature timestamp to be earlier than the current block time

Update:

This is an issue with the Lite account.

For key page entries, UpdateKeyPage resets LastUsedOn for all entries (and increments the version), so as long as you don't do that for all the keys of the page (and higher priority pages) you're safe.

Paul and I talked about some ideas for dealing with that potential problem, but we decided for 1.0 we would tell people "don't do that"

Status:

The Accumulate team has acknowledged this issue.

[FP-9] Incomplete CLI Functions Low Acknowledged

Issue/Risk: Implementation Vulnerability

Description:

The implementation of CLI has the following issues:

1. LastUseOn isn't displayed correctly
2. An account's "threshold" and "Authorities" are not displayed
3. transaction data are not verified locally and incorrect transaction data may be submitted.

Recommendation:

Consider adding more CLI functions.

Update:

Development of the CLI functions has a relatively low priority. Currently, most of the efforts will be put in the development of the core blockchain modules. More CLI functions and features will be added in version 1.1

Status:

The Accumulate team has acknowledged this issue.

[FP-10] Missing Validation for Parameters Low ✓ Fixed

Issue/Risk: Data Security

Description:

Transactions of SyntheticBurnTokens, BurnTokens and IssueTokens didn't check whether the input amount is greater than 0.

Recommendation:

Consider adding conditional checks to ensure the input amount to be equal to or greater than 0

Update/Status:

The Accumulate team has fixed this issue.

[FP-11] Missing Validation for Parameters Low ✓ Fixed

Issue/Risk: Data Security

Description:

Transactions of CreateKeyBook and CreateKeyPage didn't check if the Key is 32 bytes.

Recommendation:

Consider adding checks to `CreateKeyPage` and `CreateKeyBook` to ensure that the hashes are actually hashes.

Update/Status:

The Accumulate team has fixed this issue.

[FP-12] Inappropriate Configurations Low ✓ Fixed

Issue/Risk: Data Security

Description:

Slowloris attacks could happen since `ReadHeaderTimeout` is not configured in the http.Server

internal/accumulated/run.go:243: d.api = &http.Server{Handler: d.jrpc.NewMux()}

Recommendation:

Consider adding a `ReadHeaderTimeout` in the above API to prevent slowloris attacks.

Update/Status:

The Accumulate team has fixed this issue.

[FP-13] Issue with Atomic Transactions Low

Acknowledged

Issue/Risk: Transaction Security

Description:

A transaction on Accumulate crosses BVNs and is executed in multiple blocks across BVNs, and each BVN maintains its own state. Therefore a transaction is not atomic and cannot be reverted.

The existing implementation to handle this issue is as follows:

1 using sequence numbers for synthetic transactions to ensure all the transactions are properly ordered.

2 when a synthetic transaction fails, sending a new transaction to revert the state.

However we are concerned this implementation may not be able to handle complex situations. We are reviewing the potential issues or possible risks when the implementation cannot handle complex situations.

Recommendation:

Consider implementing this distributed system following ACID's rules:

Atomicity: it would be better to stringently follow this

Consistency: a transactions is only allowed to apply if it satisfies all the constraints

Isolation: it is better to minimize the interference between parallel transactions

Durability: it would be better to stringently follow this

Consider applying the Actor model, for more details please refer to:

https://en.wikipedia.org/wiki/Actor_model and

How CosmWasm applies the Actor model at:

https://docs.cosmwasm.com/docs/1.0/architecture/actor/

Update:

The available operations (transactions) are carefully designed to allow us to revert failed operations. In 1.0, no single operation will be so complex that it cannot be reverted. We do intend to add atomic transaction sets in 1.1 or 1.2. Those will need to be designed very carefully.

Status:

The Accumulate team has acknowledged this issue.

[FP-14] Oracle Security Low Acknowledged

Issue/Risk: Oracle Security

Description:

The ACME's price is firstly fed by an operator and then verified by other operators. This single price feed cannot ensure the security of the price feed.

Recommendation:

Consider supplying the prices by DN's validators and calculating a final price based on the prices.

Update:

The Accumulate team plans to implement a more secure oracle in a new version.

Status:

The Accumulate team has acknowledged this issue.

[FP-15] Inappropriate Validator Power Low Acknowledged

Issue/Risk: Security in Consensus Algorithm

Description:

In ABCI's `EndBlock`, pendingUpdates are returned at every block, and the validators information is updated in the function willChangeGlobals. The Power of the newly added validators is always 1.

Recommendation:

Consider using a DPOS algorithm to calculate the actual power of a validator.

Update/Status:

The Accumulate team has acknowledged this issue.

[FP-16] Inappropriate Characters Low Acknowledged

Issue/Risk: Misc

Description:

The existing Accumulate implementation supports W3C URL and UTF-8 coding. Therefore users can use non-English words such as Chinese, emoji and space to create account names (for instance “acc://fp.acme/h哈哈”). However these names are not user-friendly and users are easily making mistakes when typing these names.

Recommendation:

Consider only allowing readable ASCII characters to be used in names and if W3C URL is supported, consider using a uniform standard to Encode and Decode. However an encoded URL is not user-friendly and is in general too long for users to easily and correctly type.

Update:

The Accumulate team is testing and assessing the risks of using Unicode characters to create URLs. The Accumulate team is considering restricting characters to be used to create URLs. Final decisions are to be made.

Status:

The Accumulate team has acknowledged this issue.

[FP-17] Signature Schemes Could be Improved Low

Acknowledged

Issue/Risk: Code Improvement

Description:

“SyntheticForwardTransaction” is a special Synthetic transaction. It forwards a user signature. The existing implementation forwards the following three kinds of signatures

a) A key (normal) signature needs to be forwarded if the signer is not local to (does not belong to the same root identity as) the principal

b) A delegated signature needs to be forwarded if the delegatee and delegator are remote (not local) or the delegator and principal are remote

c) SignatureSet was used in multisig scenarios to forward a collection of signatures

This logic is too complex to understand and handle.

Recommendation:

Consider redesigning and simplifying the logic

Update:

Signatures will be properly part of the transaction flow in 1.1. We will create a transaction type for signatures and signatures in submitted envelopes will be 'promoted' to that transaction type. Thus all the tracking we do for transactions will apply to signatures.

In 1.1 we will probably forward all signatures to simplify the logic. SignatureSet is no longer used. We need to clean up the signature code, but that will be 1.1

Status:

The Accumulate team has acknowledged the issue.

[FP-18] Reinforcing Token Security Low Acknowledged

Issue/Risk: Code Improvement

Description:

Accumulate's token design may have some potential issues

The existing implementation is as follow:

a) a token issuer only keeps its token's meta data in the BVN it belongs to. The issuer doesn't keep the balance of the token holder's account

b) The balance of a token holder's account is saved in the BVN that the token holder belongs to. When the holder transfers his/her token, the BVN the holder belongs to will verify the transaction and reduce the balance after it is verified.

c) When a sender's BVN initiates a Synthetic transaction, an anchor will be sent to the DN.

d) The receiver's BVN receives the Synthetic transaction, verifies the anchor and increases the receiver's balance if everything goes well.

The issues with the implementation:

a) when a transaction is initiated, the balance of the sender is only verified in the sender's BVN but the receiver BVN cannot verify the sender's balance.

b) However the anchor can only be used to verify if a transaction is initiated but cannot be used to verify if the sender has sufficient balance for this transaction.

c) the sender BVN and receiver BVN independently verify the sender's balance and the receiver's balance. If either party's BVN were a malicious network the BVN would change its balance at will.

Recommendation:

Consider implementing the logic this way: a token issuer's BVN keeps all token holders' account balances. When a token transfer is initiated, the token issuer's BVN will act as a relayer and verify the transaction.

Update:

The State for each account are maintained by the Binary Patricia Tree (BPT) in each BVN. The state is the result of processing transactions, and not "issued" by the leader. The leader orders transactions only. As a result, the state of the BVN can only be compromised by authorities via Tenderment. We will be shuffling the authorities over the BVN (automatically in the next update).

Synthetic transactions are generated by processing transactions, and are part of the BPT in a BVN. So cryptographically proving the BPT is correct is done by every node in a BVN, and falsifying synthetic transactions is cryptographically provable.

Also, each account is audit-able independently. So cryptographic proofs of false transactions are very small.

If we look at banking, I would say that very little is validated. I have been told by bank auditors that banks are impossible to fully audit. The individual accounts only exist within a banking system, and when you factor in fractional reserve banking, the size of the money supply is effectively unbounded. But that is another story.

The purpose of the DVN is to create cryptographic proofs for all synthetic transactions, and the ability to create small fraud proofs.

The power of our design is the ability to do a full audit on any account down to the token issuer without involving the entire blockchain. This means everything gets audited more rather than trust more centralized full nodes.

The Accumulate team believes this makes the protocol harder to compromise rather than easier when compared to monolithic blockchains like Ethereum and Bitcoin.

Monolithic blockchains have a single ledger, making the whole thing valid or invalid (all transactions exist within the ledger). But as that grows, the problem of validation gets harder and harder, and fewer parties deal with validation.

Accumulate ensures that components of the blockchain can be validated independently, but also completely. That means more parties can validate parts, and the sum of all that validation is greater and decentralized.

In the next release the Accumulate team will have data servers that collect all the information from all the BVNs and the DVN. Think of it like what google does for the Internet, the information is pulled from other systems. And while nobody can process it all, they can chase and validate the links they are interested in.

Fundamentally Accumulate relies on BVNs not being evil. And if a BVN does turn evil, we need to detect it immediately. For this to work, Accumulate must ensure that no one party or cabal gains control of all the active BVN validators, and must incentivize followers to ensure any evil activity is detected as soon as possible. Accumulate will be launching with a relatively small set (~25) of trusted operators. Accumulate will have 3 BVNs, each of which will have 10 active validators, and those will be spread across the 25 operators. PoS v2 will include randomly shuffling BVN assignments and active validator assignments, which will make it much harder for a cabal to gain control of a BVN.

If evil activity occurs and is detected, confidence in the network will drop, the value of ACME will drop, and the operators' stake will drop. Given a sufficiently large pool of followers, evil activity should be detected immediately. This should provide incentive against operators trying to steal ACME. Of course it would not deter someone solely interested in hurting the network.

The branch pre-release-2 is all the changes the Accumulate team has finished so far that the Accumulate team plans to release.

There's some minor changes (<100 LOC) made for debugging on the beta testnet that the Accumulate team may want to incorporate into the mainnet. Those changes are mostly improved log messages

Status:

The Accumulate team has acknowledged this issue.

[FP-19] Limitation in Signature Algorithms Low

Acknowledged

Issue/Risk: Code Improvement

Description:

The existing implementation only supports Ed25519 based signatures.

Recommendation:

Consider supporting other signature algorithms as well.

Update:

The Accumulate team will add support in the next version.

Status:

The Accumulate team has acknowledged this issue.

[FP-20] Inappropriate Error Handling Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

It is better to use `error` rather than `panic` to handle errors in transaction submission, query etc otherwise it might suffer from DOS attacks.

Recommendation:

Consider using `error` instead of `panic`

Update/Status:

The Accumulate team has fixed this issue.

[FP-21] White-paper Not Updated Info Acknowledged

Issue/Risk: Misc

Description:

Some of the sections in the white paper are out-of-date.

Recommendation:

Consider updating the white-paper

Update/Status:

The Accumulate team has acknowledged this issue.

[FP-22] More Static Testing Needed Info Acknowledged

Issue/Risk: Misc

Description:

The existing implementation needs more static testing.

Recommendation:

Consider doing more static testing using Linter which has revive, errcheck, govet, staticcheck, typecheck, gosec and more.

Update/Status:

The Accumulate team has acknowledged this.

[FP-23] More Comments Needed Info Acknowledged

Issue/Risk: Misc

Description:

Some code implementations have complex logic. More comments are needed for them for developers and auditors to understand the implementations better.

Recommendation:

Consider adding more comments for the these code implementations.

Update/Status:

The Accumulate team has acknowledged this.

09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- N/A

Appendix

Static Testing

Accumulate Network is implemented in Golang. The GolangCI-Lint can be used to test it. Specifically some security issues could be uncovered by using gosec.

Steps to Run GolangCI-Lint:

1. Install GolangCI-Lint

```
# go install github.com/golangci/golangci-lint/cmd/golangci-lint@latest
```

2. Run Test Command

```
# golangci-lint run -v --no-config --enable-all
```

3. Check Test Results

Steps to Run gosec to Test Security:

1. Install gosec

```
# go install github.com/securego/gosec/v2/cmd/gosec@latest
```

2. Run Test Command

```
# gosec ./...
```

Manual Review

Manual review is to manually check code line by line to uncover issues or risks based on supplied documents and source code.

For more details about manual review, please refer to OWASP's "Go Programming Language Secure Coding Practices Guide"

Dynamic Testing

Unit Testing

The Accumulate team has done unit tests for most of the code. For the modules that have been found to have potential issues or risks during the manual review, more unit tests need to be done.

Fuzz Testing

Fuzz tests may be needed to test how the system react when invalid, unexpected, or random data are provided as inputs.

System Testing

More system tests are needed

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech