Blockchain Protocol Security Analysis Report

Customer: GOAT Network

Date: 08/11/2024

We express our gratitude to the GOAT Network team for the collaborative engagement that enabled the execution of this Blockchain Protocol Security Assessment.

GOAT Network is an innovative Bitcoin Layer-2 solution designed to enhance scalability and provide yield opportunities for BTC holders. By utilizing BitVM2 for off-chain computation verification, it streamlines transaction processing and promotes efficiency. The network introduces goatBTC, a BTC-pegged asset tailored for diverse applications in decentralized finance (DeFi) and traditional finance (TradFi), facilitating seamless integration within both ecosystems.

Document

Name	Blockchain Protocol Review and Security Analysis Report for GOAT Network
Audited By	Reza Mir
Approved By	Nino Lipartiia
Website	https://www.goat.network/
Changelog	04/11/2024 - Preliminary Report
Changelog	08/11/2024 - Final Report
Platform	GOAT Metwork
Language	Golang
Tags	Cosmos SDK, Consensus, L2
Methodology	https://hackenio.cc/blockchain_methodology

Review Scope

Repository	https://github.com/GOATNetwork/goat/tree/testnet-2
Commit	ce01477b3686dae5526a2e9ff9f240df3a0fa222

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

5	2	1	2
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	2
Medium	1
Low	2

Vulnerability	Severity
F-2024-6771 - Vulnerable Dependencies	High
F-2024-6928 - Locking Amount Retention for Downgraded Validator Status	High
F-2024-6934 - Execution Disruption from Non-Validator Requests	Medium
F-2024-6935 - Inconsistency in Validator Eligibility Due to Threshold Bypass	Low
F-2024-6958 - Risks Associated with Integer Overflow Conversions	Low

Documentation quality

- The documentation for the installation and build process is well-structured and accessible.
- Additional module-specific documentation would greatly enhance clarity and depth of understanding.
- While code comments are present, further elaboration in certain areas could improve comprehensibility.
- A README file is currently missing from the repository.

Code quality

- The codebase displays strong overall quality, yet several areas present opportunities for refinement.
- Certain code sections lack sufficient granularity and contain repetitive elements
- Robust error handling mechanisms contribute to system stability and make debugging more efficient.
- Increasing unit and end-to-end (e2e) test coverage is recommended to further improve reliability and maintainability.

Architecture quality

- The project is built on the robust foundation of the Cosmos SDK.
- The modular design supports ease of maintenance and facilitates future scalability.

Table of Contents

System Overview

This assessment centers on the implementation of the *x/locking* module within the GOAT Network. This module serves as a pivotal element of the consensus layer, orchestrating locking functionalities. Its primary features encompass:

- The facilitation of the locking and unlocking of validator funds
- The administration of reward claims and distributions
- The updating of validator statuses in accordance with the amounts locked and any modifications
- The generation of transactions pertaining to the locking smart contract

The logic underpinning the *x/locking* module is intricately interwoven with both the execution layer and the smart contract. Requests processed by this module emanate from the smart contract and traverse through the *x/goat* module, thereby ensuring seamless integration within the network's operational framework. Ultimately, this architecture bolsters the overall efficacy and resilience of the GOAT Network.

Risks

- **Scope Definition and Security Assumptions:** The audit does not encompass the entire repository, leaving components outside the scope that may introduce vulnerabilities, potentially jeopardizing overall security due to the interconnectedness of protocols.
- **Request Validation:** All requests processed by the locking module are assumed to originate from the locking smart contract and are preliminarily validated by both the smart contract and the execution layer.
  - Notably, sensitive requests, such as adding validators and updating the reward pool or tokens, lack rigorous governance validation and are processed as received, relying on prior checks before reaching the locking module. These validation checks are beyond the audit's scope.

Findings

Vulnerability Details

F-2024-6771 - Vulnerable Dependencies - High

Description:

A comprehensive security analysis conducted with the tools `govulncheck` and `snyk` has uncovered multiple potential vulnerabilities within libraries used by the project. These findings highlight the need for immediate attention to mitigate potential risks and ensure the continued security and stability of the system.

Standard Library:

- **A. Description**: *go/build/constraint:* stack exhaustion in `Parse`.
  - **Impact**: Calling `Parse` on a "// +build" build tag line with deeply nested expressions can cause a panic due to stack exhaustion.
  - **GitHub Advisory**: GHSA-j7vj-rw65-4v26
  - **Affected Versions**: < go 1.23.0-0
  - **Fixed in**: go 1.23.1
  - **CVSS Score**: 7.5 (High)
- **B. Description**: Stack exhaustion in `Decoder.Decode` in *encoding/gob*.
  - **Impact**: Calling `Decoder.Decode` on a message which contains deeply nested structures can cause a panic due to stack exhaustion.
  - **GitHub Advisory**: GHSA-crqm-pwhx-j97f
  - **Affected Versions**: < go 1.23.0-0
  - **Fixed in**: go 1.23.1
  - **CVSS Score**: 7.5 (High)
- **C. Description**: Stack exhaustion in all `Parse` functions in *go/parser*.
  - **Impact**: Calling any of the Parse functions on Go source code which contains deeply nested literals can cause a panic due to stack exhaustion..
  - **GitHub Advisory**: GHSA-8xfx-rj4p-23jm
  - **Affected Versions**: < go 1.23.0-0
  - **Fixed in**: go 1.23.1
  - **CVSS Score**: N/A

External vulnerable dependencies:

github.com/cometbft/cometbft : v0.38.12

- **A. Description**: Default configuration param for `Evidence` may limit window of validity

- **Impact**: A default configuration in CometBFT has been found to be small for common use cases, and may prevent the slashing mechanism from working in specific cases. The default values for `EvidenceParams.MaxAgeNumBlocks` and `EvidenceParams.MaxAgeDuration` consensus parameters may not be sufficient for common use cases to provide coverage for the entire unbonding period for a chain (`Staking.UnbondingTime`).
    - **GitHub Advisory**: [GHSA-555p-m4v6-cqxv]()
    - **Affected Versions**: All
    - **Fixed in**: None
    - **CVSS Score**: N/A (Low)
- **B. Description**: CometBFT's default for `BlockParams.MaxBytes` consensus parameter may increase block times and affect consensus participation.
    - **Impact**: This issue does not represent an actively exploitable vulnerability that would result in a direct loss of funds, however it may have a slight impact on block latency depending on a network's topography.
    - **GitHub Advisory**: [GHSA-hq58-p9mv-338c]()
    - **Affected Versions**: All
    - **Fixed in**: None
    - **CVSS Score**: N/A (Low)

[cosmos/cosmos-sdk : v0.50.10]()

- **A. Description**: The `x/crisis` package does not cause chain halt.
    - **Impact**: If an invariant check fails on a Cosmos SDK network and a transaction is sent to the `x/crisis` module to halt the chain, the chain does not halt.
    - **GitHub Advisory**: [GHSA-qfc5-6r3j-jj22]()
    - **Affected Versions**: All versions
    - **Fixed in**: None
    - **CVSS Score**: N/A (Low)
- **B. Description**: The `x/crisis` package does not charge `ConstantFee`.
    - **Impact**: If a transaction is sent to the `x/crisis` module to check an invariant, the `ConstantFee` parameter of the chain is NOT charged.
    - **GitHub Advisory**: [GHSA-w5w5-2882-47pc]()
    - **Affected Versions**: All versions
    - **Fixed in**: None
    - **CVSS Score**: N/A (Low)

[github.com/golang/protobuf v1.5.4]()

- **A. Description**: Deprecated: Use the "[google.golang.org/protobuf]()" module instead.

[github.com/gin-gonic/gin : v1.7.0]()

- **A. Description**: Inconsistent Interpretation of HTTP Requests
  - **Impact**: When *gin* is exposed directly to the internet, a client's IP can be spoofed by setting the X-Forwarded-For header.
  - **GitHub Advisory**: [GHSA-h395-qcrw-5vmq]()
  - **Affected Versions**: < 1.7.7
  - **Fixed in**: 1.7.7
  - **CVSS Score**: 7.1 (High)
- **B. Description**: Improper input validation.
  - **Impact**: gin before version 1.9.0 are vulnerable to Improper Input Validation by allowing an attacker to use a specially crafted request via the X-Forwarded-Prefix header, potentially leading to cache poisoning. Although this issue does not pose a significant threat on its own it can serve as an input vector for other more impactful vulnerabilities. However, successful exploitation may depend on the server configuration and whether the header is used in the application logic.
  - **GitHub Advisory**: [GHSA-3vp4-m3rf-835h]()
  - **Affected Versions**: < 1.9.0
  - **Fixed in**: 1.9.0
  - **CVSS Score**: 5.6 (Medium)
- **C. Description**: Gin does not properly sanitize filename parameter of `Context.FileAttachment` function.
  - **Impact**: The filename parameter of the `Context.FileAttachment` function is not properly sanitized. A maliciously crafted filename can cause the Content-Disposition header to be sent with an unexpected filename value or otherwise modify the Content-Disposition header. For example, a filename of "setup.bat";x=.txt" will be sent as a file named "setup.bat".
  - **GitHub Advisory**: [GHSA-2c4m-59x9-fr2g]()
  - **Affected Versions**: >= 1.3.1-0.20190301021747-ccb9e902956d, < 1.9.1
  - **Fixed in**: 1.9.1
  - **CVSS Score**: 4.3 (Medium)

Assets:

- Dependencies

Status: Fixed

Classification

Impact: 4/5

Likelihood: 3/5

Severity: High

Recommendations

Remediation:

To ensure the security and stability of the project, it is crucial to address the vulnerabilities identified in its dependencies.

While updating all dependencies to their latest versions is ideal, the following minimum versions are required to address the identified vulnerabilities:

- **Upgrade minimum Go version:**
  - change `go 1.22` to `go 1.23.1`
- **Direct vulnerable dependencies:**
  - *gin-gonic/gin*: v1.9.1
- **Eliminate deprecated decencies:**
  - *github.com/golang/protobuf*

Implementing these recommendations will significantly enhance the project's defense against known vulnerabilities, ensuring a more secure and stable environment moving forward.

F-2024-6928 - Locking Amount Retention for Downgraded Validator Status - High

Description:

The coins locked by each validator are stored in both the `Keeper.Locking` map and the `Locking` field within each `Validator` in `Keeper.Validators`, as shown below:

```
Keeper struct {
...
// (token,validator) => locking, it's used for updating power when
the token weight is updated
Locking collections.Map[collections.Pair[string, sdktypes.ConsAddre
ss], math.Int]
...
Validators collections.Map[sdktypes.ConsAddress, types.Validator]
}

type Validator struct {
// the total locking
Locking github_com_cosmos_cosmos_sdk_types.Coins `protobuf:"bytes,3
,rep,name=locking,proto3,castrepeated=github.com/cosmos/cosmos-sdk/
types.Coins" json:"locking"`
...
}
```

The `Locking` field in each `Validator` represents the total coins held by the validator, regardless of its status. Conversely, `Keeper.Locking` only tracks locked amounts for `Active` and `Pending` validators, as these statuses are relevant for participation in consensus. When a validator's status changes to `Downgrade`, `Inactive`, or `Tombstoned`, the respective locked amounts are removed from `Keeper.Locking`.

However, an issue exists within the `Unlock` function: if called for a `Downgrade` status validator, it modifies `Keeper.Locking` based on `validator.Locking` without verifying the validator's current status. Refer to the Evidence section for more information.

This issue could be further compounded if the `UpdateTokenWeightRequest` is invoked. Specifically, when `onWeightChanged` updates power for `Active` and `Pending` validators, it iterates over `Keeper.Locking`, potentially including downgraded validators. Consequently, this update could introduce inconsistencies into both `Keeper.Locking` and `Keeper.PowerRanking`, as `Keeper.PowerRanking` is recalculated using this incorrect data.

These inconsistencies could significantly hinder network stability and integrity, as they may disrupt the accurate selection of validators, potentially impacting consensus operations and the overall reliability of the blockchain.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 5/5

Severity: High

Recommendations

Remediation: To address this issue, it is recommended to incorporate a conditional check within the `Unlock` function to ensure that any modifications to the `Keeper.Locking` map are contingent on the validator's current status. Specifically, this check should verify that only validators with an `Active` or `Pending` status can trigger updates to `Keeper.Locking`. For validators with a `Downgrade`, `Inactive`, or `Tombstoned` status, the `Unlock` function should bypass adjustments to `Keeper.Locking`.

Implementing this check would ensure that `Keeper.Locking` accurately reflects only the locking amounts relevant to active participants in the consensus process, thus maintaining consistency and preventing unauthorized modifications by validators no longer eligible for consensus participation.

Evidences

Testing Unlock execution for the types.Downgrade validator status

Reproduce:

```
func (suite *KeeperTestSuite) TestUnlockForDowngrade() {
// Initial setup
now := time.Now().UTC()
newctx := suite.Context.WithBlockTime(now)

for address, token := range suite.Token {
err := suite.Keeper.Tokens.Set(newctx, address, token)
suite.Require().NoError(err)
}

err := suite.Keeper.Threshold.Set(newctx, types.Threshold{List: sui
te.Threshold})
suite.Require().NoError(err)

// Adding downgraded validator
var Validator = types.Validator{
Pubkey: common.Hex2Bytes("02269670f94151626e04ecf71792cf7ff89bd5bd4
0b07a408c2a6a11e10085ef76"),
Reward: math.ZeroInt(),
GasReward: math.ZeroInt(),
Status: types.Downgrade,
JailedUntil: now.Add(time.Hour),
Locking: sdk.NewCoins(
sdk.NewCoin(TestTokenDenom, math.NewIntFromUint64(1e19+1e18)),
),
}

Address := sdk.ConsAddress(common.Hex2Bytes("d466208fb91604fc161707
12ad186caa34c9b005"))
Recipient := common.HexToAddress("0xf8cd0c0a6f6568a7bf015134ffc5850
cb9f53a57")

err = suite.Keeper.Validators.Set(newctx, Address, Validator)
suite.Require().NoError(err)

// Create unlocking request
reqs := []*goattypes.UnlockRequest{
{Id: 3, Validator: common.Address(Address), Recipient: Recipient, T
oken: TestToken, Amount: new(big.Int).SetUint64(1e19)},
}

// The Keeper.Locking is empty before the Unlock call
lockingIter, err := suite.Keeper.Locking.Iterate(newctx, nil)
suite.Require().NoError(err)
suite.Require().False(lockingIter.Valid())

// Calling Unlock
err = suite.Keeper.Unlock(newctx, reqs)
suite.Require().NoError(err)

// The Keeper.Locking state is updated, it contains the downgraded
validator
lockingIter, err = suite.Keeper.Locking.Iterate(newctx, nil)
suite.Require().NoError(err)

kv, err := lockingIter.KeyValue()
suite.Require().NoError(err)

realAddress := string(kv.Key.K2())
```

See more

Results:

```
PASS
ok github.com/goatnetwork/goat/x/locking/keeper 0.027s
```

F-2024-6934 - Execution Disruption from Non-Validator Requests - Medium

Description:

The `locking` module is designed to process validator-related requests for managing locked funds. Each method in the Keeper processes a batch of requests of a specific type, as shown in the `Lock` method definition:

```
func (k Keeper) Lock(ctx context.Context, reqs []*goattypes.LockReq
uest) error
```

The issue arises in how invalid requests are handled. If a batch includes a request to lock funds from an address not listed as a validator, an error is returned immediately, discarding all other valid requests in the batch:

```
validator, err := k.Validators.Get(sdkctx, valdtAddr)
if err != nil {
return err
}
```

This opens an attack vector: a non-validator address can repeatedly send locking requests, effectively blocking the processing of legitimate requests. This vulnerability could significantly impact the blockchain's state, as it disrupts the execution of valid locking requests.

This issue affects multiple request types:

- `Lock`
- `Unlock`
- `Claim`
- `Create`

For the `Create` request, the attack pattern differs. A validator can submit a request to create an already existing validator, which triggers an error and blocks other valid `Create` requests in the batch.

Although additional validation layers (such as on the execution layer or within smart contracts) could help mitigate this issue, they fall outside the scope of this assessment. Robust handling in the consensus layer is recommended to mitigate potential risks and to adhere to a defense-in-depth strategy, ensuring that even if other layers have validation flaws, the consensus layer effectively mitigates potential disruptions.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status:	Mitigated

Classification

Impact:	4/5
Likelihood:	2/5
Severity:	Medium

Recommendations

Remediation:

It is recommended to enhance the error-handling mechanism to address this vulnerability by ignoring invalid requests rather than halting batch processing. This approach could include:

- Skipping the execution of invalid requests without terminating the transaction.
- Logging malicious addresses to track suspicious activity.
- Returning a comprehensive error report at the end of the batch execution, detailing any invalid requests without blocking valid ones.

Implementing these changes will help maintain blockchain stability and improve the resilience of the validator locking mechanism against disruptive attacks.

Resolution:

The issue is considered mitigated. The GOAT team provided the following comment:

> Not issue, all requests have been checked in EVM.
>
> 1. original locking request,
> https://github.com/GOATNetwork/goat-contracts/blob/main/contracts/locking/Locking.sol#L140
> 2. EVM handles locking request,
> https://github.com/ethereum/go-ethereum/compare/master...GOATNetwork:goat-geth:testnet-2#diff-689874b30f6f905ce6c47cdefeddcf052b467a7936a981f4b6cf38939e10d924R130

Verification of this mechanism is beyond the scope of the current audit.

F-2024-6935 - Inconsistency in Validator Eligibility Due to Threshold Bypass - Low

Description: The current implementation of the `Lock` and `Unlock` functions presumes that a validator is eligible to participate in the validation process and vie for active status only if its locked amount satisfies the threshold specified by the `Keeper`. Specifically, when a validator's locked balance falls below this threshold following an unlock operation, it is automatically exited from the validator set, with its status marked as `Inactive`:

```
isExiting := validator.Status == types.Inactive ||
validator.Status == types.Tombstoned || lockingAmount.LT(token.Thre
shold)
```

However, a discrepancy arises in the validator update process, as the threshold is overlooked in the `EndBlocker` logic, where only the `PowerRanking` metric determines the next validator set.

As a result, a validator may circumvent the threshold requirement by increasing its `PowerRanking` through locking a substantial amount of a single coin, disregarding the threshold criteria across other tokens. This enables a validator to achieve inclusion in the validator set solely based on `PowerRanking`, even if the threshold criterion is unmet.

This inconsistency may lead to unexpected validator inclusion or exclusion, potential exploitation of the threshold rule, and erosion of trust in the validator selection mechanism.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Mitigated

Classification

Impact: 2/5

Likelihood: 2/5

Severity: Low

Recommendations

Remediation: To address this inconsistency, it is strongly recommended to:

- Incorporate a threshold verification step within the `EndBlocker` process to ensure compliance with the established threshold.
- Supplement the documentation with clear guidelines on the expected behavior and the threshold's role in determining validator eligibility.

These steps will reinforce adherence to the threshold requirement and support the integrity and reliability of the validator selection process.

Resolution: The issue is considered mitigated. The GOAT team provided the following comment:

> Not issue, all requests are from Smart contract and are handled in a deterministic order.
>
> A validator must meet the threshold when it’s approved to be alive. https://github.com/GOATNetwork/goat-contracts/blob/testnet-2/contracts/locking/Locking.sol#L497

Verification of this mechanism is beyond the scope of the current audit.

F-2024-6958 - Risks Associated with Integer Overflow Conversions - Low

Description:

Multiple integer overflow risks during type conversions were identified through security scanning using `Gosec`. Among these, the following three present significant security concerns:

```
[goat/x/locking/keeper/keeper.go:143] - G115 (CWE-190): integer ove
rflow conversion uint64 -> int64 (Confidence: MEDIUM, Severity: HIG
H)
142: PubKey: cmsecp256k1.PubKey(validator.Pubkey),
> 143: Power: int64(kv.Value),
144: Name: types.ValidatorName(kv.Key.Bytes()),

Autofix:

[goat/x/locking/keeper/abci.go:108] - G115 (CWE-190): integer overf
low conversion uint64 -> int64 (Confidence: MEDIUM, Severity: HIGH)
107: newSet = append(newSet, abci.ValidatorUpdate{
> 108: Power: int64(validator.Power), PubKey: validator.CMPubkey()}
)
109: k.Logger().Info("Validator set updated", "address", types.Vali
datorName(valAddr), "power", validator.Power)

Autofix:

[goat/x/locking/keeper/abci.go:91] - G115 (CWE-190): integer overfl
ow conversion uint64 -> int64 (Confidence: MEDIUM, Severity: HIGH)
90: newSet = append(newSet, abci.ValidatorUpdate{
> 91: Power: int64(validator.Power), PubKey: validator.CMPubkey()})
92: k.Logger().Info("Validator set updated", "address", types.Valid
atorName(valAddr), "power", validator.Power)
```

See Go Language Specification: Numeric Types.

While the likelihood of triggering an arithmetic overflow in these conversions is minimal—due to the necessity of having a large number of tokens or an excessively large weight value—the potential impact on the system could be significant. This risk is heightened by the fact that some of these conversions occur within the `EndBlocker` implementation, where overflows could have a more substantial effect.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Accepted

Classification

Impact: 4/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: To mitigate the risk of integer overflow, particularly during conversions between different integer types, the following measures are essential:

- **Implement Robust Range Checks**: Prior to converting a `uint64` value to an `int64`, verify that the value falls within the valid range of `int64`. This can be accomplished through conditional checks to validate the size of the number before conversion.
- **Utilize Safe Arithmetic Libraries**: Consider using libraries or built-in functions that automatically handle overflow detection and provide safe arithmetic operations.
- **Conduct Regular Code Reviews and Testing**: Regularly test your code with edge cases to ensure that your range checks are effective and comprehensive.

Incorporating these practices will help prevent unexpected behaviors and safeguard the integrity of your application’s operations.

Resolution: The issue has been accepted with the following comment from the GOAT team:

> Not issue, the Int64 is enough to represent all the locked asset, including BTC/wBTC etc.

We agree that the likelihood of overflow is very low.

Observation Details

F-2024-6909 - Code Quality Deficiencies - Info

Description: As part of our ongoing audit of the Golang codebase, several opportunities for improvement have been identified to enhance code style, readability, and maintainability. Although the code functions correctly, these recommendations aim to streamline development practices, facilitating easier management and extension of the codebase over time.

Key Areas for Improvement:

- Descriptive Variable Names
- Ensuring Up-to-Date Software in Docker and GitHub Workflows
- Execute Tests Using Makefile Commands
- Proofreading
- Redundant Alias
- Unused Functions and Variables

For further details and examples, please refer to the **Evidence** section.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Accepted

Recommendations

Remediation: To elevate code quality and enforce rigorous standards within your team, the following recommendations are advised:

- **Address Existing Warnings:** It is important to resolve the current warnings related to code style, correctness, and quality identified in the analysis. This foundational step will eliminate inconsistencies and enhance the overall health of the codebase.
- **Integrate Static Analysis Tools:** While `golangci-lint` effectively identifies many common issues, incorporating additional static analysis tools can further enhance code quality and detect potential bugs and vulnerabilities. Consider the following tools:
  - Staticcheck
  - Revive
  - **GoLand IDE**: Utilizing **GoLand** as the development environment has proven beneficial in identifying various issues during code review.

- **Establish Style Guidelines:** Develop or adopt a comprehensive Go style guide. Utilize official recommendations or community guides to ensure consistency and readability across the codebase. Encourage all developers to adhere to these guidelines diligently.
- **Implement Code Reviews:** Adopt a rigorous code review process to identify errors, enforce coding standards, and promote best practices. Regular code reviews will contribute significantly to maintaining a high-quality codebase.
- **Enhance Continuous Integration:** Integrate static analysis tools into your continuous integration (CI) pipeline to facilitate early detection of linting issues during the development process, ensuring code quality before deployment.

By implementing these measures, the codebase will be fortified against potential issues, ensuring the consistent delivery of robust and reliable software.

Resolution: The GOAT team has accepted the issue with the following comment:

> We will update in the next few releases

Evidences

Descriptive Variable Names

Issue details: Using more descriptive variable names improves code readability. Replacing vague identifiers like lastSet or newSet helps maintain clarity, especially in larger or more complex functions.

Ensuring Up-to-Date Software in Docker and GitHub Workflows

Issue details: To ensure optimal security and performance in your development environment, it is crucial to regularly review and update your DockerFile and GitHub Action workflows. This involves verifying that you are utilizing the latest software updates and dependencies. For instance, using the most recent version of the Golang runtime can help mitigate potential vulnerabilities and enhance the efficiency of your applications. By keeping your software components up-to-date, you not only improve system stability but also align with best practices in software development and deployment.

Execute Tests Using Makefile Commands

Location: https://github.com/GOATNetwork/goat/blob/0918cafcd8085ce08cc6e71beeff5c01ee1e91ca/Makefile#L106

Issue details: There is no command to run test files inside the `Makefile`.

`install`, `run-tests`, and `test-e2e` are not defined yet.

Proofreading

Issue details:

1. recipent
2. GoatToekenDenom
3. caculation
4. caculate
5. happend
6. waitting
7. rewrad
8. duation
9. paritial
10. duation
11. interator

Redundant Alias

Issue details: Reports aliases of imported packages that may be omitted. Usually, such aliases equal to the names of the imported packages, so aliases have no effect and one can use package names directly.

1. goattypes

Unused Functions and Variables

Issue details:

1. KeyPrefix
2. param *types.Params

F-2024-6919 - Test Failures and Insufficient Coverage - Info

Description:

Upon executing `go test ./x/locking/...`, one of the tests in the *x/locking* module is currently failing:

```
--- FAIL: TestGenesisState_Validate (0.00s)
--- FAIL: TestGenesisState_Validate/valid_genesis_state (0.00s)
genesis_test.go:35:
Error Trace: /home/nino/projects/goat/x/locking/types/genesis_test.
go:35
Error: Received unexpected error:
zero signed window values
Test: TestGenesisState_Validate/valid_genesis_state
```

Moreover, running test coverage analysis with:

```
go test -coverprofile=coverage ./x/locking/...
go tool cover -func coverage
```

reveals that while the *x/locking/keeper* directory achieves 73.6% coverage, many other files fall significantly short, resulting in an average of only 11.9% coverage for the entire locking module. These coverage gaps underscore potential vulnerabilities and areas where undetected bugs or inconsistencies may arise.

High test coverage is crucial to ensure the *x/locking* module's reliability and security, detecting issues early and preventing bugs from affecting core network functions. Especially for modules that impact validator states and consensus, thorough testing reduces the risk of unforeseen errors or vulnerabilities in production.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Accepted

Recommendations

Remediation:

To address the failing tests in the `locking` package, begin by analyzing the test outputs to identify specific causes of failure. Resolve any incorrect code or logical inconsistencies that deviate from expected behavior, and expand test cases to cover all relevant scenarios, including edge cases. Once corrections are made, rerun tests to verify the issues are resolved.

Additionally, the following steps are recommended to enhance test coverage and overall code reliability:

- **Increase Test Coverage:** Aim for at least 80% coverage, prioritizing critical components to ensure core functionality is rigorously tested.

- **Develop Comprehensive Unit Tests:** Design unit tests that cover a broad range of cases, including edge cases, to ensure functions behave as expected.
- **Adopt Test-Driven Development (TDD):** Encourage TDD practices to clarify requirements and improve code robustness through early test planning.
- **Leverage Code Coverage Tools:** Use these tools to identify and address untested areas, focusing additional testing efforts on these parts of the code.
- **Add Integration and End-to-End Tests:** Develop integration and E2E tests to verify interactions between components, capturing issues that unit tests alone may overlook.
- **Integrate Testing into the CI Pipeline:** Automate test execution within the CI pipeline to ensure continuous quality checks and immediate feedback on code changes.

By implementing these steps, the `locking` package can achieve more reliable operation, reduced vulnerability to bugs, and improved maintainability over time.

Resolution: The GOAT team has accepted the issue with the following comment:

> We will update in the next few releases

[F-2024-6946](#) - Streamlining Coin Locking Restrictions for Disengaged Validator States - Info

Description: The current system allows for the locking of tokens for validators in both `Inactive` and `Tombstone` states. While this action does not affect the validation selection process or alter the network's total power, it raises questions about its necessity and practicality.

`Inactive` and `Tombstone` validators cannot participate in the network as active validators. Therefore, allowing them to increase their locked balance serves no functional purpose and may be considered redundant. Since these validators cannot transition back to an active state, any additional funds locked do not enhance their operational capabilities or influence their status within the network.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Accepted

Recommendations

Remediation: While adding tokens to a validator's locked balance in `Tombstoned` or `Inactive` states poses no security risks, it is an unnecessary operation that could be optimized for improved system efficiency. Restricting the ability to increase the locked balance for validators in these states would prevent redundant actions, contributing to a cleaner and more streamlined process.

To implement this change, review the conditions under which locking coins to a validator’s balance is permitted. If adding funds in the `Tombstoned` or `Inactive` states does not align with intended functionality, enforce conditions that allow token locking only for validators in active or relevant states.

In addition, update documentation to clearly outline the expected behavior of the system, specifying acceptable states for locking actions and the rationale behind restricting balance updates for inactive validators. This will ensure transparency and understanding for both the development team and all blockchain participants, fostering greater alignment on system functionality and intended use.

Resolution: The GOAT team has accepted the issue with the following comment:

> We will update in the next few releases

F-2024-6956 - Incorporating All Validator States in Event Emissions - Info

Description: Currently, there are two events related to validator status changes:

```
EventTypeDowngraded = "validator_downgraded"
EventTypeTombstoned = "validator_tombstoned"
```

The current implementation effectively captures changes to a validator's status, specifically regarding `Downgrade` and `Tombstone` states. However, to enhance the system's robustness and ensure comprehensive coverage, it is essential to include additional potential states that a validator may encounter, such as `Pending`, `Active`, and `Inactive`.

In files like *msg_lock.go* and *msg_unlock.go*, the validator's status is updated, as demonstrated in the following examples:

- validator.Status = types.Pending
- validator.Status = types.Inactive

Publishing relevant events for each status change is advisable, as this practice keeps the system informed and responsive to updates. This proactive approach will enhance overall system functionality and ensure that all validator states are appropriately managed.

Assets:

- Custom Module Locking [https://github.com/GOATNetwork/goat]

Status: Accepted

Recommendations

Remediation: It is crucial to include event handling for additional validator states, such as changes to `Pending`, `Active`, and `Inactive`. This enhancement will improve monitoring and management of validator lifecycle events.

In files like *msg_lock.go* and *msg_unlock.go*, utilize the `EventManager().EmitEvent` function for each status change. This ensures that all system components are promptly informed of updates, enhancing overall responsiveness and tracking capabilities.

Resolution: The GOAT team has accepted the issue with the following comment:

> We will update in the next few releases

Disclaimers

Hacken Disclaimer

The blockchain protocol given for audit has been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in the protocol source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other protocol statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the blockchain protocol.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Blockchain protocols are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the protocol can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited blockchain protocol.

Appendix 1. Severity Definitions

Severity	Description
Critical	Vulnerabilities that can lead to a complete breakdown of the blockchain network's security, privacy, integrity, or availability fall under this category. They can disrupt the consensus mechanism, enabling a malicious entity to take control of the majority of nodes or facilitate 51% attacks. In addition, issues that could lead to widespread crashing of nodes, leading to a complete breakdown or significant halt of the network, are also considered critical along with issues that can lead to a massive theft of assets. Immediate attention and mitigation are required.
High	High severity vulnerabilities are those that do not immediately risk the complete security or integrity of the network but can cause substantial harm. These are issues that could cause the crashing of several nodes, leading to temporary disruption of the network, or could manipulate the consensus mechanism to a certain extent, but not enough to execute a 51% attack. Partial breaches of privacy, unauthorized but limited access to sensitive information, and affecting the reliable execution of smart contracts also fall under this category.
Medium	Medium severity vulnerabilities could negatively affect the blockchain protocol but are usually not capable of causing catastrophic damage. These could include vulnerabilities that allow minor breaches of user privacy, can slow down transaction processing, or can lead to relatively small financial losses. It may be possible to exploit these vulnerabilities under specific circumstances, or they may require a high level of access to exploit effectively.
Low	Low severity vulnerabilities are minor flaws in the blockchain protocol that might not have a direct impact on security but could cause minor inefficiencies in transaction processing or slight delays in block propagation. They might include vulnerabilities that allow attackers to cause nuisance-level disruptions or are only exploitable under extremely rare and specific conditions. These vulnerabilities should be corrected but do not represent an immediate threat to the system.

Appendix 2. Scope

The scope of the project includes the following components from the provided repository:

Scope Details	
Repository	https://github.com/GOATNetwork/goat/tree/testnet-2
Commit	ce01477b3686dae5526a2e9ff9f240df3a0fa222
Whitepaper	https://drive.google.com/file/d/1ytrY_Q3UQbguYGruTSk20zV1QusCojRx/view

Components in Scope

The audit centers on a single module contained within the *x/locking* directory.