// HALBORN

Haqq - Vesting

Cosmos Security Assessment

Prepared by: **Halborn**

Date of Engagement: **July 3rd, 2023 - July 18th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	07/17/2023
0.2	Document Updates	07/17/2023
0.3	Draft Review	07/18/2023
1.0	Remediation Plan	12/03/2023
1.1	Remediation Plan Review	12/04/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Haqq engaged Halborn to conduct a security assessment on their modules beginning on July 3rd, 2023 and ending on July 18th, 2023. The security assessment was scoped to the modules provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to assessment the security of the module. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Verify the security of the Cosmos modules: vesting.
- Ensure that module functions operate as intended.
- Identify potential security issues with the modules.

In summary, Halborn identified some security risks that were addressed and accepted by the Haqq team.

1.3 SCOPE

The assessment focused primarily on the custom implementation of the Vesting module located at x/vesting/ in the codebase.

1. IN-SCOPE TREE & COMMIT :

Commit ID : a7158c7355a863735e3b3596d21dd96cc235fae8

REMEDIATION COMMIT ID :

- c02701d35eacb32fdf7d0d1ae1749a7e2e8b7fa5

1.4 TEST APPROACH & METHODOLOGY

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

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

2. RISK METHODOLOGY

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

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

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

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

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

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

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

Attack Complexity (AX):

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

Metrics:

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

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

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

Integrity (I):

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

Availability (A):

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

Deposit (D):

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

Yield (Y):

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

Metrics:

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

Impact I is calculated using the following formula:

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

2.3 SEVERITY COEFFICIENT

Reversibility (R):

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

Scope (S):

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

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

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

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

The score is rounded up to 1 decimal places.

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

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	2	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) PROJECT VULNERABLE TO BARBERRY VULNERABILITY	Critical (9.1)	SOLVED - 12/01/2023
(HAL-02) LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT	Low (2.1)	RISK ACCEPTED
(HAL-03) USE OF VULNERABLE DEPENDENCIES	Low (4.4)	SOLVED - 12/01/2023
(HAL-04) TODOS IN CODEBASE	Informational (0.0)	ACKNOWLEDGED
(HAL-05) LACK OF CHECK FOR PERIOD AMOUNT VALUE IN MSGCONVERTINTOVESTINGACCOUNT FUNCTION	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) PROJECT VULNERABLE TO BARBERRY VULNERABILITY - CRITICAL(9.1)

Description:

The project is vulnerable to the barberry security issue. Cosmos versions less than 0.46.13 and 0.47.3 are affected. This issue can lead to a chain halt, and so it is considered to have a critical impact.

Code Location:

go.mod#L8

Listing 1

```
1 module github.com/haqq-network/haqq
2
3 go 1.19
4
5 require (
6     cosmossdk.io/errors v1.0.0-beta.7
7     cosmossdk.io/math v1.0.0-beta.6
8     github.com/cosmos/cosmos-sdk v0.46.10
```

BVSS:

AO:A/AC:M/AX:M/C:N/I:C/A:C/D:H/Y:H/R:N/S:C (9.1)

Recommendation:

Apply the latest Cosmos security updates. Monitor the Cosmos releases and official forums to stay informed of security issues.

Remediation Plan:

SOLVED: The Haqq team solved the issue by upgrading the Cosmos SDK.

Commit ID: c02701d35eacb32fdf7d0d1ae1749a7e2e8b7fa5

4.2 (HAL-02) LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT - LOW (2.1)

Description:

The **Haqq Chain** system lacks comprehensive CosmosSDK simulations and invariants for its **coinomics** module. More complete use of the simulation feature would make it easier to fuzz test the entire blockchain and help ensure that invariants hold.

BVSS:

AO:A/AC:M/AX:L/C:N/I:M/A:N/D:N/Y:N/R:P/S:C (2.1)

Recommendation:

Eventually, extend the simulation module to cover all operations that can occur in a real Haqq Chain deployment, along with all possible error states, and run it many times before each release. Make sure of the following:

- All module operations are included in the simulation module.
- The simulation uses some accounts (e.g., between 5 and 20) to increase the likelihood of an interesting state change.
- The simulation uses the currencies/tokens that will be used in the production network.
- The simulation continues to run when a transaction fails.
- All paths of the transaction code are executed. (Enable code coverage to see how often individual lines are executed.)

Remediation Plan:

RISK ACCEPTED: The Haqq team accepted the risk of this finding.

4.3 (HAL-03) USE OF VULNERABLE DEPENDENCIES - LOW (4.4)

Description:

A variety of vulnerabilities exists in dependencies used by the project's vesting module.

Code Location:

Vulnerabilities flagged by the tool nancy:

ID	Package	Rating	Description
CVE-2021-42219	go-ethereum	HIGH	Uncontrolled Resource Consumption
CVE-2022-23328	go-ethereum	HIGH	Uncontrolled Resource Consumption
CVE-2022-37450	go-ethereum	MEDIUM	Improper Input Validation

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:L/Y:L/R:N/S:U (4.4)

Recommendation:

Where possible, keep dependencies patched in order to reduce the risk of the system being attacked using known vulnerabilities. It is recommended that the Haqq team runs the nancy and govulncheck, tools regularly and fix as many warnings as possible.

Remediation Plan:

SOLVED: The Haqq team solved the issue by upgrading the vulnerable dependencies.

Commit ID: c02701d35eacb32fdf7d0d1ae1749a7e2e8b7fa5

4.4 (HAL-04) TODOS IN CODEBASE - INFORMATIONAL (0.0)

Description:

Numerous code comments in the codebase contain TODO messages or other developer notes indicating malfunctioning or missing functionality.

Code Location:

Listing 2

```
./types/schedule.go:167:// TODO: rename and add comprehensive
↳ comments, this is currently not maintainable
```

BVSS:

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

Recommendation:

It is recommended to use a separate issue tracker or other task management software to track bugs and features rather than using code comments. Developer notes in comments are very likely to be overlooked and to become out of date relative to the code.

Remediation Plan:

ACKNOWLEDGED: The Haqq team acknowledged the risk of this finding.

4.5 (HAL-05) LACK OF CHECK FOR PERIOD AMOUNT VALUE IN MSGCONVERTINTOVESTINGACCOUNT FUNCTION - INFORMATIONAL (0.0)

Description:

In the MsgConvertIntoVestingAccount function, there is a validation check for weather period.Length is greater than 0, both in msg.LockupPeriods and msg.VestingPeriods. However, there seems to be no check to ensure that period.Amount is greater than 0. The lack of this check means that a vesting period or lockup period could potentially be created with an amount of 0, which might not be the desired behavior.

Code Location:

/x/vesting/types/msg.go#L288

Listing 3

```
20 func (msg MsgConvertIntoVestingAccount) ValidateBasic() error {
21 ..
22     vestingCoins := sdk.NewCoins()
23     for i, period := range msg.VestingPeriods {
24         if period.Length < 1 {
25             return errorsmod.Wrapf(errortypes.ErrInvalidRequest, "
↳ invalid period length of %d in period %d, length must be greater
↳ than 0", period.Length, i)
26         }
27         vestingCoins = vestingCoins.Add(period.Amount...)
28     }
29
30 ...
31 }
```

BVSS:

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

Recommendation:

It is recommended to include a check to ensure that `period.Amount` is greater than `0`. Here is an example of how this could be done:

Listing 4

```
for i, period := range msg.LockupPeriods {
    if period.Length < 1 {
        return errorsmod.Wrapf(errortypes.ErrInvalidRequest, "
↳ invalid period length of %d in period %d, length must be greater
↳ than 0", period.Length, i)
    }
    if period.Amount.IsZero() {
        return errorsmod.Wrapf(errortypes.ErrInvalidRequest, "
↳ invalid period amount of %s in period %d, amount must be greater
↳ than 0", period.Amount, i)
    }
    lockupCoins = lockupCoins.Add(period.Amount...)
}

for i, period := range msg.VestingPeriods {
    if period.Length < 1 {
        return errorsmod.Wrapf(errortypes.ErrInvalidRequest, "
↳ invalid period length of %d in period %d, length must be greater
↳ than 0", period.Length, i)
    }
    if period.Amount.IsZero() {
        return errorsmod.Wrapf(errortypes.ErrInvalidRequest, "
↳ invalid period amount of %s in period %d, amount must be greater
↳ than 0", period.Amount, i)
    }
    vestingCoins = vestingCoins.Add(period.Amount...)
}
```

Remediation Plan:

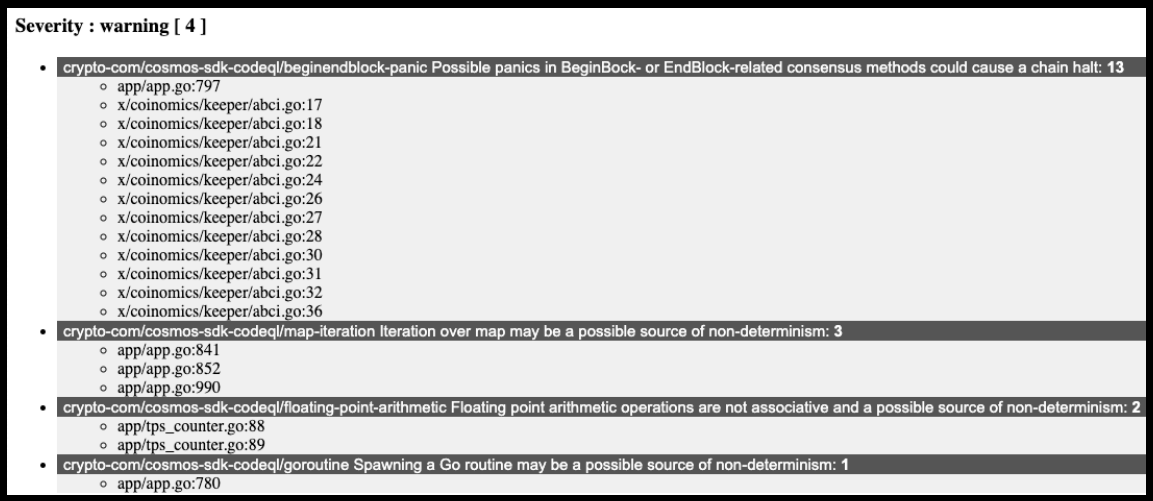
ACKNOWLEDGED: The Haqq team acknowledged the risk of this finding.

AUTOMATED TESTING

5.1 Automated Testing -- Overview

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

5.2 CodeQL



Severity : warning [4]

- crypto-com/cosmos-sdk-codeql/beginendblock-panic Possible panics in BeginBock- or EndBlock-related consensus methods could cause a chain halt: **13**
  - app/app.go:797
  - x/coinomics/keeper/abci.go:17
  - x/coinomics/keeper/abci.go:18
  - x/coinomics/keeper/abci.go:21
  - x/coinomics/keeper/abci.go:22
  - x/coinomics/keeper/abci.go:24
  - x/coinomics/keeper/abci.go:26
  - x/coinomics/keeper/abci.go:27
  - x/coinomics/keeper/abci.go:28
  - x/coinomics/keeper/abci.go:30
  - x/coinomics/keeper/abci.go:31
  - x/coinomics/keeper/abci.go:32
  - x/coinomics/keeper/abci.go:36
- crypto-com/cosmos-sdk-codeql/map-iteration Iteration over map may be a possible source of non-determinism: **3**
  - app/app.go:841
  - app/app.go:852
  - app/app.go:990
- crypto-com/cosmos-sdk-codeql/floating-point-arithmetic Floating point arithmetic operations are not associative and a possible source of non-determinism: **2**
  - app/tps_counter.go:88
  - app/tps_counter.go:89
- crypto-com/cosmos-sdk-codeql/goroutine Spawning a Go routine may be a possible source of non-determinism: **1**
  - app/app.go:780

Figure 1: CodeQL results

5.3 gosec

The following as an excerpt from running the tool gosec:

```
File: gosec

Results:


[/Users/user/Documents/halborn/projects/haqq/src/app/app.go:852-854] - G705 (CWE-): expected either an append, delete, or copy to another map in a range with a map (Confidence: MEDIUM, Severity: HIGH)
    851:    blockedAddrs := make(map[string]bool)
  > 852:    for acc := range maccPerms {
  > 853:        blockedAddrs[authtypes.NewModuleAddress(acc).String()] = !allowedReceivingModAcc[acc]
  > 854:    }
    855:


[/Users/user/Documents/halborn/projects/haqq/src/app/app.go:841-843] - G705 (CWE-): expected either an append, delete, or copy to another map in a range with a map (Confidence: MEDIUM, Severity: HIGH)
    840:    modAccAddrs := make(map[string]bool)
  > 841:    for acc := range maccPerms {
  > 842:        modAccAddrs[authtypes.NewModuleAddress(acc).String()] = true
  > 843:    }
    844:


[/Users/user/Documents/halborn/projects/haqq/src/x/coinomics/keeper/inflation.go:75] - G701 (CWE-): Potential integer overflow by integer type conversion (Confidence: MEDIUM, Severity: HIGH)
    74:
  > 75:     totalMintOnBlockInt := eraTargetMint.Amount.Quo(sdk.NewInt(int64(params.BlocksPerEra)))
    76:     totalMintOnBlockCoin := sdk.NewCoin(params.MintDenom, totalMintOnBlockInt)


[/Users/user/Documents/halborn/projects/haqq/src/x/coinomics/keeper/abci.go:16] - G701 (CWE-): Potential integer overflow by integer type conversion (Confidence: MEDIUM, Severity: HIGH)
    15:
  > 16:     currentBlock := uint64(ctx.BlockHeight())
    17:     currentEra := k.GetEra(ctx)


[/Users/user/Documents/halborn/projects/haqq/src/app/upgrades/v1.0.2/upgrades.go:58] - G701 (CWE-): Potential integer overflow by integer type conversion (Confidence: MEDIUM, Severity: HIGH)
    57:     // part1 * (10^18)
  > 58:     expectedSupply.Mul(part1, big.NewInt(int64(math.Pow(10, 18))))
    59:


[/Users/user/Documents/halborn/projects/haqq/src/app/upgrades/v1.0.2/upgrades.go:56] - G701 (CWE-): Potential integer overflow by integer type conversion (Confidence: MEDIUM, Severity: HIGH)
    55:     // 20*(10^9)
  > 56:     part1 := new(big.Int).Mul(big.NewInt(20), big.NewInt(int64(math.Pow(10, 9))))
    57:     // part1 * (10^18)
```

Figure 2: gosec excerpt

THANK YOU FOR CHOOSING

// HALBORN