Haqq - Coinomics

Cosmos Security Assessment

Prepared by: **Halborn**

Date of Engagement: **May 4th, 2023 - May 15th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/10/2023	John Saigle
0.2	Document Updates	05/15/2023	John Saigle
0.3	Draft Review	05/15/2023	Gokberk Gulgun
0.4	Draft Review	05/16/2023	Gabi Urrutia
1.0	Remediation Plan	07/13/2023	Gokberk Gulgun
1.1	Remediation Plan Review	07/18/2023	Gabi Urrutia

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
Gokberk.Gulgun	Halborn	Gokberk.Gulgun@halborn.com
John.saigle	Halborn	John.Saigle@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Haqq engaged Halborn to conduct a security assessment on their smart contracts beginning on May 4th, 2023 and ending on May 15th, 2023. The security assessment was scoped to the smart contracts 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 smart contract. 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 coinomics module.
- Ensure that the module functions according to the project's design.
- Examine that dependencies are up-to-date.

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

1.3 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.
- Evaluating implementation of features relative to project design documents (e.g. whitepaper)
- Ensuring correctness of the codebase.

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

2.4 SCOPE

The x/coinomics module of the Haqq codebase was in scope for this assessment.

- Haqq: https://github.com/haqq-network/haqq

The commit hash used for the assessment was as follows:

- 6741ddc47f00f13d417735ccef107c5d47b28e15

The module makes extensive use of fixed values in its calculations that are initialized during genesis. Interactions with governance (e.g x/gov) were not in scope for this assessment.

2. FIX BRANCH & COMMIT IDs:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	0	1	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) COINS ARE TRANSFERRED EVEN IF MINTING FAILS	Critical (10)	SOLVED - 06/30/2023
(HAL-02) ERRORS RETURNED BY MintAndAllocateInflation ARE IGNORED	High (8.4)	SOLVED - 07/17/2023
(HAL-03) LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT	Low (2.1)	RISK ACCEPTED
(HAL-04) POSSIBLE DIVISION BY ZERO IN CalcInflation FUNCTION	Informational (1.0)	SOLVED - 07/17/2023
(HAL-05) POSSIBLE DIVISION BY ZERO IN MintAndAllocationInflation FUNCTION	Informational (1.0)	SOLVED - 07/17/2023
(HAL-06) INTEGER OVERFLOW COULD CAUSE CHAIN HALT	Informational (1.0)	SOLVED - 07/17/2023
(HAL-07) ABCI METHOD CAN BE OPTIMIZED	Informational (0.0)	SOLVED - 07/17/2023
(HAL-08) CalcTargetMintForEra CAN BE OPTIMIZED	Informational (0.0)	SOLVED - 07/17/2023
(HAL-09) TELEMETRY MODULE NOT UTILIZED IN ENDBLOCKER FUNCTION OF COSMOS SDK	Informational (0.0)	SOLVED - 07/17/2023
(HAL-10) SPELLING MISTAKES IN THE CODEBASE	Informational (0.0)	SOLVED - 07/17/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) COINS ARE TRANSFERRED EVEN IF MINTING FAILS - CRITICAL(10)

Description:

If an error occurs when minting coins, the module's execution does not stop or process the error. Instead, the error is logged and execution continues.

Minting is immediately followed by a coin transfer. This could lead to unexpected results if coin minting fails and a transfer is attempted regardless.

Code Location:

Listing 1: Even when MintCoins returns err, SendCoinsFromModuleToModule executes (Lines 80,84)

```
71 func (k Keeper) MintAndAllocateInflation(ctx sdk.Context) error {
72     params := k.GetParams(ctx)
73     eraTargetMint := k.GetEraTargetMint(ctx)
74
75     totalMintOnBlockInt := eraTargetMint.Amount.Quo(sdk.NewInt(
 ↳ int64(params.BlocksPerEra)))
76     totalMintOnBlockCoin := sdk.NewCoin(params.MintDenom,
 ↳ totalMintOnBlockInt)
77
78     // Mint coins to coinomics module
79     if err := k.MintCoins(ctx, totalMintOnBlockCoin); err != nil {
80         ctx.Logger().Error("FAILED MintCoins: ", err.Error())
81     }
82
83     // Allocate remaining coinomics module balance to destribution
84     err := k.bankKeeper.SendCoinsFromModuleToModule(
85         ctx,
86         types.ModuleName,
87         k.feeCollectorName,
88         sdk.NewCoins(totalMintOnBlockCoin),
89     )
90     if err != nil {
        return err
    }

    return nil
}
```

Proof-of-concept:

- MintCoins is triggered but returns an error
- A transfer is attempted with SendCoinsFromModuleToModule
- Logical or accounting errors occur because coins have been transferred without a mint.

BVSS:

AO:A/AC:L/AX:M/C:N/I:C/A:H/D:N/Y:M/R:N/S:C (10)

Recommendation:

Handle errors correctly when they occur. Abandon code that has side effects if all preconditions are not met.

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing necessity checks.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.2 (HAL-02) ERRORS RETURNED BY MintAndAllocateInflation ARE IGNORED - HIGH (8.4)

Description:

The codebase does not check whether a function returns an error. This could lead to unexpected behavior, as the code will continue executing without responding to the error. This error occurs in the context of an EndBlocker function and could affect chain consensus as a result.

This compounds the effect of finding HAL-01 as the err resulting from SendCoinsFromModuleToModule is effectively responsible for surfacing errors that arise in MintCoins.

Code Location:

Listing 2: The function can return an error in multiple places. (Line 91)

```
71 func (k Keeper) MintAndAllocateInflation(ctx sdk.Context) error {
72     params := k.GetParams(ctx)
73     eraTargetMint := k.GetEraTargetMint(ctx)
74
75     totalMintOnBlockInt := eraTargetMint.Amount.Quo(sdk.NewInt(
↳ int64(params.BlocksPerEra)))
76     totalMintOnBlockCoin := sdk.NewCoin(params.MintDenom,
↳ totalMintOnBlockInt)
77
78     // Mint coins to coinomics module
79     if err := k.MintCoins(ctx, totalMintOnBlockCoin); err != nil {
80         ctx.Logger().Error("FAILED MintCoins: ", err.Error())
81     }
82
83     // Allocate remaining coinomics module balance to destribution
84     err := k.bankKeeper.SendCoinsFromModuleToModule(
85         ctx,
86         types.ModuleName,
87         k.feeCollectorName,
88         sdk.NewCoins(totalMintOnBlockCoin),
89     )
90     if err != nil {
91         return err
92     }
93 
94     return nil
95 }
```

Listing 3: The error is ignored (Line 36)

```
 8 func (k Keeper) EndBlocker(ctx sdk.Context) {
 9     params := k.GetParams(ctx)
10 
11     // NOTE: ignore end of block if coinomics is disabled
12     if !params.EnableCoinomics {
13         return
14     }
15 
16     currentBlock := uint64(ctx.BlockHeight())
17     currentEra := k.GetEra(ctx)
18     eraForBlock := k.CountEraForBlock(ctx, params, currentEra,
   ↳ currentBlock)
19 
20     if currentEra != eraForBlock {
21         k.SetEra(ctx, eraForBlock)
22         k.SetEraStartedAtBlock(ctx, currentBlock)
23 
24         nextEraTargetMint := k.CalcTargetMintForEra(ctx,
   ↳ eraForBlock)
25 
26         currentTotalSupply := k.bankKeeper.GetSupply(ctx, types.
   ↳ DefaultMintDenom)
27         nextEraClosingSupply := currentTotalSupply.AddAmount(
   ↳ nextEraTargetMint.Amount)
28         nextEraInflation := k.CalcInflation(ctx, eraForBlock,
   ↳ nextEraClosingSupply, nextEraTargetMint)
29 
30         k.SetEraTargetMint(ctx, nextEraTargetMint)
31         k.SetEraClosingSupply(ctx, nextEraClosingSupply)
32         k.SetInflation(ctx, nextEraInflation)
33     }
34 
```

```
    //nolint:errcheck
    k.MintAndAllocateInflation(ctx)
}
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:H/A:M/D:N/Y:M/R:N/S:C (8.4)

Recommendation:

Any function that can return an error should have its result checked and emit an Event that can be logged and responded to by the system.

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing necessity checks.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.3 (HAL-03) 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 the issue.

4.4 (HAL-04) POSSIBLE DIVISION BY ZERO IN CalcInflation FUNCTION - INFORMATIONAL (1.0)

Description:

There is a possible division by zero in a consensus-related function. This could lead to a panic and a chain halt.

Code Location:

The coinomics module contains a file abci.go. The function EndBlocker calls the function CalcInflation in x/module/keeper/inflation.go.

CalcInflation performs division via the Quo function without first checking that the divisor is not equal to 0.

Listing 4: The argument to Quo is unchecked

```go
func (k Keeper) CalcInflation(ctx sdk.Context, era uint64, eraTargetSupply sdk.Coin, eraTargetMint sdk.Coin) sdk.Dec {
    if era > 50 {
        return sdk.NewDec(0)
    }

    return sdk.NewDecFromInt(eraTargetMint.Amount).
        Quo(sdk.NewDecFromInt(eraTargetSupply.SubAmount(eraTargetMint.Amount).Amount)).
        Mul(sdk.NewDec(100))
}
```

x/module/keeper/abci.go

Listing 5: CalcInflation is called within a consensus method.

```go
    if currentEra != eraForBlock {
        k.SetEra(ctx, eraForBlock)
        k.SetEraStartedAtBlock(ctx, currentBlock)

        nextEraTargetMint := k.CalcTargetMintForEra(ctx, eraForBlock)

        currentTotalSupply := k.bankKeeper.GetSupply(ctx, types.DefaultMintDenom)
        nextEraClosingSupply := currentTotalSupply.AddAmount(nextEraTargetMint.Amount)
        nextEraInflation := k.CalcInflation(ctx, eraForBlock, nextEraClosingSupply, nextEraTargetMint)

        k.SetEraTargetMint(ctx, nextEraTargetMint)
        k.SetEraClosingSupply(ctx, nextEraClosingSupply)
        k.SetInflation(ctx, nextEraInflation)
    }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:N/R:F/S:C (1.0)

Recommendation:

Add checks to ensure that no division by zero is possible. It is important to note that the values used in these calculations are configured at genesis and there is no way for an attacker to modify them. As a result, the likelihood of this issue occurring is very low.

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing necessity checks.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.5 (HAL-05) POSSIBLE DIVISION BY ZERO IN MintAndAllocationInflation FUNCTION - INFORMATIONAL (1.0)

Description:

There is a possible division by zero in a consensus-related function. This could lead to a panic and a chain halt.

Code Location:

MintAndAllocateInflation performs division via the Quo function without first checking that the divisor is not equal to 0. When this value was used during manual testing, no blocks per produced.

Listing 6

```
func (k Keeper) MintAndAllocateInflation(ctx sdk.Context) error {
    params := k.GetParams(ctx)
    eraTargetMint := k.GetEraTargetMint(ctx)

    totalMintOnBlockInt := eraTargetMint.Amount.Quo(sdk.NewInt(int64(params.BlocksPerEra)))
    totalMintOnBlockCoin := sdk.NewCoin(params.MintDenom, totalMintOnBlockInt)

    // Mint coins to coinomics module
    if err := k.MintCoins(ctx, totalMintOnBlockCoin); err != nil {
        ctx.Logger().Error("FAILED MintCoins: ", err.Error())
    }

    // Allocate remaining coinomics module balance to destribution
    err := k.bankKeeper.SendCoinsFromModuleToModule(
        ctx,
        types.ModuleName,
        k.feeCollectorName,
        sdk.NewCoins(totalMintOnBlockCoin),
    )
    if err != nil {
        return err
    }

    return nil
}
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:N/R:F/S:C (1.0)

Recommendation:

Add checks to ensure that no division by zero is possible. It is important to note that the value used in this calculation is configured at genesis and there is no way for an attacker to modify it. As a result, the likelihood of this issue occurring is very low.

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing necessity checks.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.6 (HAL-06) INTEGER OVERFLOW COULD CAUSE CHAIN HALT - INFORMATIONAL (1.0)

Description:

There is a risk of integer overflow occurring when a parameter is converted from an unsigned integer to a signer integer. If an overflow occurs, the value will be converted from a very large, positive number to a negative number. When this value is passed as an argument to NewCoin a panic occurs. This calculation happens in the context of a consensus function, so if an overflow occurs, the chain will halt.

Code Location:

params.BlocksPerEra is an unsigned integer and is converted via the int64() function to a signed integer. For values greater than MAX_UINT64, an overflow will occur.

The MintAndAllocateInflation is called by the EndBlocker function in x/coinomics/keepr/abci.go. Therefore, if NewCoin panics due to processing an argument that is a negative number, the chain will halt.

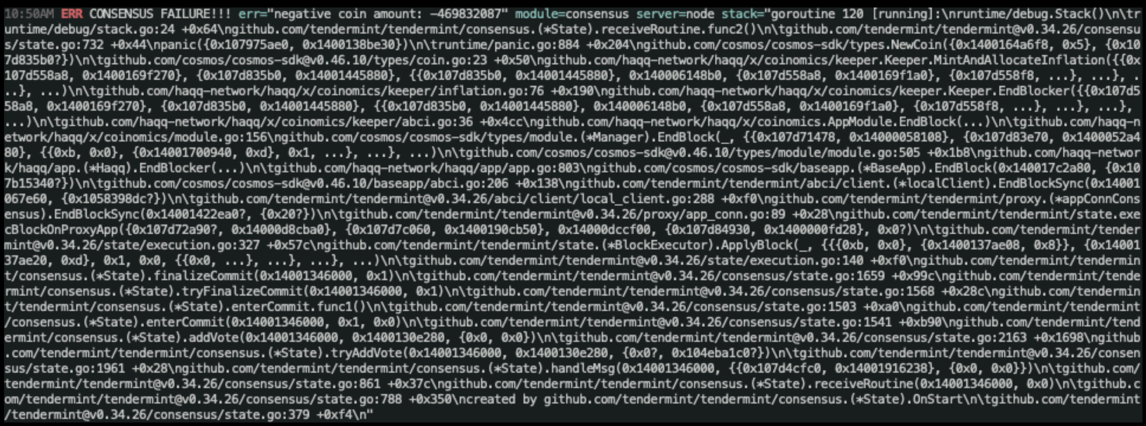
Code Location:

Listing 7

```go
func (k Keeper) MintAndAllocateInflation(ctx sdk.Context) error {
    params := k.GetParams(ctx)
    eraTargetMint := k.GetEraTargetMint(ctx)

    totalMintOnBlockInt := eraTargetMint.Amount.Quo(sdk.NewInt(int64(params.BlocksPerEra)))

    Additionally, there is a case where a function can return an error but the error is not checked by the calling code.
    totalMintOnBlockCoin := sdk.NewCoin(params.MintDenom,
↳ totalMintOnBlockInt)

    // Mint coins to coinomics module
    if err := k.MintCoins(ctx, totalMintOnBlockCoin); err != nil {
        ctx.Logger().Error("FAILED MintCoins: ", err.Error())
    }

    // Allocate remaining coinomics module balance to destribution
    err := k.bankKeeper.SendCoinsFromModuleToModule(
        ctx,
        types.ModuleName,
        k.feeCollectorName,
        sdk.NewCoins(totalMintOnBlockCoin),
    )
    if err != nil {
        return err
    }

    return nil
}
```

Proof-of-concept

- Set blocksPerEra == math.MaxInt64 + 1.
- Run the chain.
- A consensus error occurs due to a negative coin amount (see screenshot).



```
10:50AM ERR CONSENSUS FAILURE!!! err="negative coin amount: -469832087" module=consensus server=node stack="goroutine 120 [running]:\nruntime/debug.Stack()\n\tr
untime/debug/stack.go:24 +0x64\ngithub.com/tendermint/tendermint/consensus.(*State).receiveRoutine.func2()\n\tgithub.com/tendermint/tendermint@v0.34.26/consensu
s/state.go:732 +0x44\npanic({0x107975ae0, 0x1400138be30})\n\truntime/panic.go:884 +0x204\ngithub.com/cosmos/cosmos-sdk/types.NewCoin({0x1400164a6f8, 0x5}, {0x10
7d835b0?})\n\tgithub.com/cosmos/cosmos-sdk@v0.46.10/types/coin.go:23 +0x50\ngithub.com/haqq-network/haqq/x/coinomics/keeper.Keeper.MintAndAllocateInflation({{0x
107d558a8, 0x1400169f270}, {0x107d835b0, 0x14001445880}, {{0x107d835b0, 0x14001445880}, 0x140006148b0, {0x107d558a8, 0x1400169f1a0}, {0x107d558f8, ...}, ...}, .
..}, ...)\n\tgithub.com/haqq-network/haqq/x/coinomics/keeper/inflation.go:76 +0x190\ngithub.com/haqq-network/haqq/x/coinomics/keeper.Keeper.EndBlocker({{0x107d5
58a8, 0x1400169f270}, {0x107d835b0, 0x14001445880}, {{0x107d835b0, 0x14001445880}, 0x140006148b0, {0x107d558a8, 0x1400169f1a0}, {0x107d558f8, ...}, ...}, ...},
...)\n\tgithub.com/haqq-network/haqq/x/coinomics/keeper/abci.go:36 +0x4cc\ngithub.com/haqq-network/haqq/x/coinomics.AppModule.EndBlock(...)\n\tgithub.com/haqq-n
etwork/haqq/x/coinomics/module.go:156\ngithub.com/cosmos/cosmos-sdk/types/module.(*Manager).EndBlock(_, {{0x107d71478, 0x14000058108}, {0x107d83e70, 0x1400052a4
80}, {{0xb, 0x0}, {0x14001700940, 0xd}, 0x1, ...}, ...}, ...)\n\tgithub.com/cosmos/cosmos-sdk@v0.46.10/types/module/module.go:505 +0x1b8\ngithub.com/haqq-networ
k/haqq/app.(*Haqq).EndBlocker(...)\n\tgithub.com/haqq-network/haqq/app/app.go:803\ngithub.com/cosmos/cosmos-sdk/baseapp.(*BaseApp).EndBlock(0x140017c2a80, {0x10
7b15340?})\n\tgithub.com/cosmos/cosmos-sdk@v0.46.10/baseapp/abci.go:206 +0x138\ngithub.com/tendermint/tendermint/abci/client.(*localClient).EndBlockSync(0x14001
067e60, {0x1058398dc?})\n\tgithub.com/tendermint/tendermint@v0.34.26/abci/client/local_client.go:288 +0xf0\ngithub.com/tendermint/tendermint/proxy.(*appConnCons
ensus).EndBlockSync(0x14001422ea0?, {0x20?})\n\tgithub.com/tendermint/tendermint@v0.34.26/proxy/app_conn.go:89 +0x28\ngithub.com/tendermint/tendermint/state.exe
cBlockOnProxyApp({0x107d72a90?, 0x14000d8cba0}, {0x107d7c060, 0x1400190cb50}, 0x14000dccf00, {0x107d84930, 0x1400000fd28}, 0x0?)\n\tgithub.com/tendermint/tender
mint@v0.34.26/state/execution.go:327 +0x57c\ngithub.com/tendermint/tendermint/state.(*BlockExecutor).ApplyBlock(_, {{{0xb, 0x0}, {0x1400137ae08, 0x8}}, {0x14001
37ae20, 0xd}, 0x1, 0x0, {{0x0, ...}, ...}, ...}, ...)\n\tgithub.com/tendermint/tendermint@v0.34.26/state/execution.go:140 +0xf0\ngithub.com/tendermint/tendermin
t/consensus.(*State).finalizeCommit(0x14001346000, 0x1)\n\tgithub.com/tendermint/tendermint@v0.34.26/consensus/state.go:1659 +0x99c\ngithub.com/tendermint/tende
rmint/consensus.(*State).tryFinalizeCommit(0x14001346000, 0x1)\n\tgithub.com/tendermint/tendermint@v0.34.26/consensus/state.go:1568 +0x28c\ngithub.com/tendermin
t/tendermint/consensus.(*State).enterCommit.func1()\n\tgithub.com/tendermint/tendermint@v0.34.26/consensus/state.go:1503 +0xa0\ngithub.com/tendermint/tendermint
/consensus.(*State).enterCommit(0x14001346000, 0x1, 0x0)\n\tgithub.com/tendermint/tendermint@v0.34.26/consensus/state.go:1541 +0xb90\ngithub.com/tendermint/tend
ermint/consensus.(*State).addVote(0x14001346000, 0x1400130e280, {0x0, 0x0})\n\tgithub.com/tendermint/tendermint@v0.34.26/consensus/state.go:2163 +0x1698\ngithub
.com/tendermint/tendermint/consensus.(*State).tryAddVote(0x14001346000, 0x1400130e280, {0x0?, 0x104eba1c0?})\n\tgithub.com/tendermint/tendermint@v0.34.26/consen
sus/state.go:1961 +0x28\ngithub.com/tendermint/tendermint/consensus.(*State).handleMsg(0x14001346000, {{0x107d4cfc0, 0x14001916238}, {0x0, 0x0}})\n\tgithub.com/
tendermint/tendermint@v0.34.26/consensus/state.go:861 +0x37c\ngithub.com/tendermint/tendermint/consensus.(*State).receiveRoutine(0x14001346000, 0x0)\n\tgithub.c
om/tendermint/tendermint@v0.34.26/consensus/state.go:788 +0x350\ncreated by github.com/tendermint/tendermint/consensus.(*State).OnStart\n\tgithub.com/tendermint
/tendermint@v0.34.26/consensus/state.go:379 +0xf4\n"
```

Figure 1: The chain panics due to an issue in NewCoin

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:N/R:F/S:C (1.0)

Recommendation:

Add validation to ensure that params.BlocksPerEra is always greater than 0 and less than MAX_INT64 in order to avoid a chain halt. It is important to note that this value is configured during consensus and there is no method for an attacker to modify it. As a result, the likelihood of this issue occurring is very low.

Remediation Plan:

SOLVED: The Haqq team solved the issue by implementing necessity checks.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.7 (HAL-07) ABCI METHOD CAN BE OPTIMIZED - INFORMATIONAL (0.0)

Description:

There are unnecessary calculations performed in a consensus method. If these are eliminated, the computational load of producing blocks will be reduced.

Code Location:

According to the design of the protocol and its inflation mechanism, when eraNumber is greater than 50, CalcTargetMintForEra and CalcInflation will always return 0. For further information on this, refer to the project's whitepaper and the calculations in the file x/coinomics/keeper/inflation.go.

For this reason, the following functions can be skipped as they will never change, i.e., EraTargetMint and Inflation will always be 0 and EraClosingSupply will always be MaxSupply.

Therefore, the calculations can be skipped, which can reduce computational load on validators.

x/coinomics/keeper/inflation.go

Listing 8

```go
    if currentEra != eraForBlock {
        k.SetEra(ctx, eraForBlock)
        k.SetEraStartedAtBlock(ctx, currentBlock)

        nextEraTargetMint := k.CalcTargetMintForEra(ctx, eraForBlock)

        currentTotalSupply := k.bankKeeper.GetSupply(ctx, types.DefaultMintDenom)
        nextEraClosingSupply := currentTotalSupply.AddAmount(
↳ nextEraTargetMint.Amount)
        nextEraInflation := k.CalcInflation(ctx, eraForBlock,
↳ nextEraClosingSupply, nextEraTargetMint)

        k.SetEraTargetMint(ctx, nextEraTargetMint)
        k.SetEraClosingSupply(ctx, nextEraClosingSupply)
        k.SetInflation(ctx, nextEraInflation)
    }
```

BVSS:

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

Recommendation:

A check can be added in this file to skip calculations after the era exceeds 50:

Listing 9

```
if currentEra > 50 {
    return
}
```

Remediation Plan:

SOLVED: The Haqq team solved the issue by optimizing code.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.8 (HAL-08) CalcTargetMintForEra CAN BE OPTIMIZED - INFORMATIONAL (0.0)

Description:

Some variables are calculated at the end of every block, but their values never change. Using a constant instead of calculating these unchanging values will save computation power for all validators.

Code Location:

The values eraPeriod and eraCoef are fixed values (2 and 0.95, respectively). For this reason, they can be set to constant values. This improves code maintainability and reduces computation.

The value den is also effectively a constant equal to 1 - 0.95^(50) and it can also be stored as a constant.

Listing 10

```
func (k Keeper) CalcTargetMintForEra(ctx sdk.Context, eraNumber uint64) sdk.Coin {
    params := k.GetParams(ctx)

    eraCoef := sdk.NewDecWithPrec(95, 2) // 0.95

    if eraNumber == 1 {
        eraPeriod := uint64(2) // 2 years
        currentTotalSupply := k.bankKeeper.GetSupply(ctx, types.DefaultMintDenom)
        maxSupply := k.GetMaxSupply(ctx)

        totalMintNeeded := maxSupply.SubAmount(currentTotalSupply.Amount)

        // ----------- NUM ------------- / ---------- DEN -------------
        // (1-era_coef)*total_mint_needed / (1-era_coef^(100/
era_period))
        num := (sdk.OneDec().Sub(eraCoef)).Mul(sdk.NewDecFromInt(totalMintNeeded.Amount))
        den := sdk.OneDec().Sub(eraCoef.Power(100 / eraPeriod))

        target := num.Quo(den)

        return sdk.NewCoin(params.MintDenom, target.RoundInt())
```

BVSS:

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

Recommendation:

Refactor all unchanging values to be constants in order to reduce unnecessary computation. This will save gas and computation time for all validators for every block produced.

Remediation Plan:

SOLVED: The Haqq team solved the issue by optimizing code.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.9 (HAL-09) TELEMETRY MODULE NOT UTILIZED IN ENDBLOCKER FUNCTION OF COSMOS SDK - INFORMATIONAL (0.0)

Description:

The function EndBlocker is responsible for performing actions at the end of each block. However, there is no telemetry measure implemented to monitor the performance and execution of these actions.

Code Location:

x/coinomics/keeper/abci.go

Listing 11

```go
func (k Keeper) EndBlocker(ctx sdk.Context) {
    params := k.GetParams(ctx)

    // NOTE: ignore end of block if coinomics is disabled
    if !params.EnableCoinomics {
        return
    }

    currentBlock := uint64(ctx.BlockHeight())
    currentEra := k.GetEra(ctx)
    eraForBlock := k.CountEraForBlock(ctx, params, currentEra, currentBlock)

    if currentEra != eraForBlock {
        k.SetEra(ctx, eraForBlock)
        k.SetEraStartedAtBlock(ctx, currentBlock)

        nextEraTargetMint := k.CalcTargetMintForEra(ctx, eraForBlock)

        currentTotalSupply := k.bankKeeper.GetSupply(ctx, types.DefaultMintDenom)
        nextEraClosingSupply := currentTotalSupply.AddAmount(nextEraTargetMint.Amount)
        nextEraInflation := k.CalcInflation(ctx, eraForBlock, nextEraClosingSupply, nextEraTargetMint)

        k.SetEraTargetMint(ctx, nextEraTargetMint)
        k.SetEraClosingSupply(ctx, nextEraClosingSupply)
        k.SetInflation(ctx, nextEraInflation)
    }

    //nolint:errcheck
    k.MintAndAllocateInflation(ctx)
}
```

BVSS:

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

Recommendation:

In order to improve the visibility and observability of the EndBlocker function in the Cosmos SDK, we recommend the integration of the telemetry module. To enable this telemetry feature, please make sure telemetry.enabled = true is set in the app.toml config file.

Listing 12

```
func (k Keeper) EndBlocker(ctx sdk.Context) {
    defer telemetry.ModuleMeasureSince(types.ModuleName, time.Now(), telemetry.MetricKeyEndBlocker)

    params := k.GetParams(ctx)

    if !params.EnableCoinomics {
        return
    }

    currentBlock := uint64(ctx.BlockHeight())
    currentEra := k.GetEra(ctx)
    eraForBlock := k.CountEraForBlock(ctx, params, currentEra, currentBlock)

    if currentEra != eraForBlock {
        k.SetEra(ctx, eraForBlock)
        k.SetEraStartedAtBlock(ctx, currentBlock)

        nextEraTargetMint := k.CalcTargetMintForEra(ctx, eraForBlock)

        currentTotalSupply := k.bankKeeper.GetSupply(ctx, types.DefaultMintDenom)
        nextEraClosingSupply := currentTotalSupply.AddAmount(nextEraTargetMint.Amount)
        nextEraInflation := k.CalcInflation(ctx, eraForBlock, nextEraClosingSupply, nextEraTargetMint)

        k.SetEraTargetMint(ctx, nextEraTargetMint)
        k.SetEraClosingSupply(ctx, nextClosingSupply)
        k.SetInflation(ctx, nextEraInflation)
    }

    k.MintAndAllocateInflation(ctx)
}
```

Remediation Plan:

SOLVED: The Haqq team solved the issue by adding telemetry into the code base.

Commit ID:

- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

4.10 (HAL-10) SPELLING MISTAKES IN THE CODEBASE - INFORMATIONAL (0.0)

Description:

The codebase contains spelling mistakes.

Code Location:

‘Target’ is spelled as ‘Traget’ in several variable names and error messages:

x/coinomics/keeper/erainfo.go

Listing 13

```go
func (k Keeper) GetEraTargetMint(ctx sdk.Context) sdk.Coin {
    params := k.GetParams(ctx)

    store := ctx.KVStore(k.storeKey)
    bz := store.Get(types.KetPrefixEraTargetMint)
    if len(bz) == 0 {
        return sdk.NewCoin(params.MintDenom, sdk.ZeroInt())
    }

    var eraTragetMintValue sdk.Coin
    err := eraTragetMintValue.Unmarshal(bz)
    if err != nil {
        panic(fmt.Errorf("unable to unmarshal eraTragetMintValue value: %w", err))
    }

    return eraTragetMintValue
}

func (k Keeper) SetEraTargetMint(ctx sdk.Context, eraMint sdk.Coin) {
    binaryEraTragetMintValue, err := eraMint.Marshal()
    if err != nil {
        panic(fmt.Errorf("unable to marshal amount value: %w", err
↳ ))
    }

    store := ctx.KVStore(k.storeKey)
    store.Set(types.KetPrefixEraTargetMint,
↳ binaryEraTragetMintValue)
}
```

BVSS:

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

Recommendation:

Fix spelling mistakes. This can help convey a sense of professionalism to project stakeholders.

Remediation Plan:

SOLVED: The Haqq team solved the issue by fixing the typo.

Commit ID:

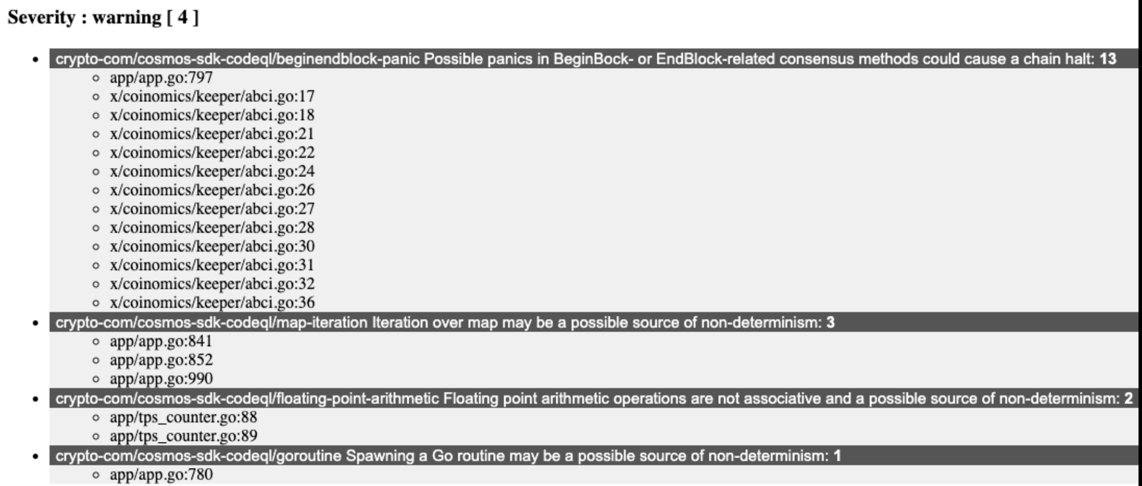
- 2a3adaac1bb240a92519b554d47cf9fe6580a4bd

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



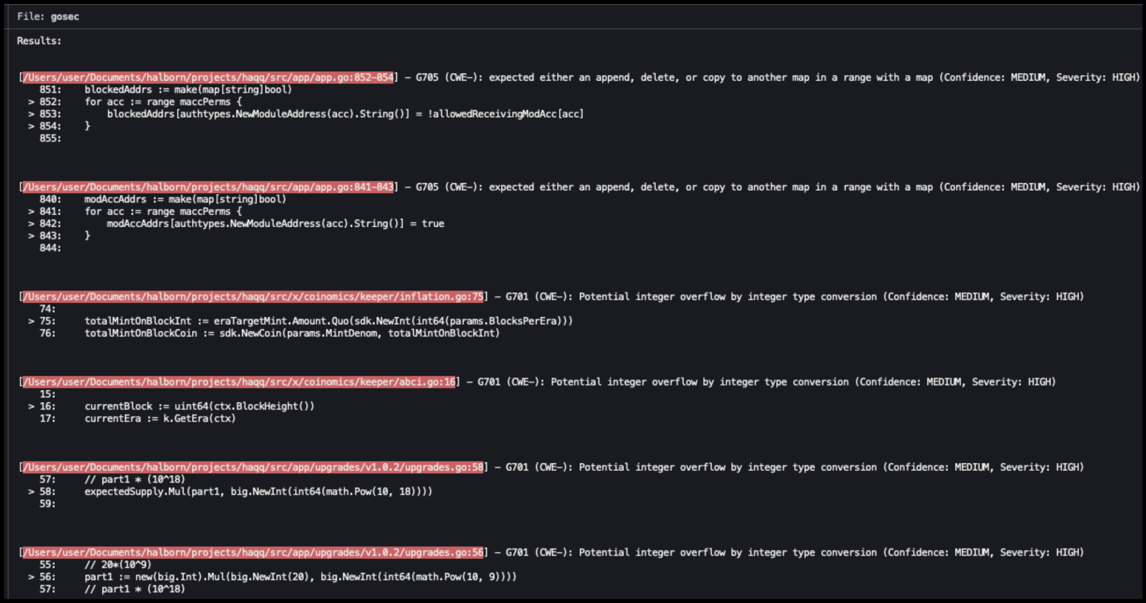
Severity : warning [4]

- 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 2: CodeQL results

5.3 gosec

The following as an excerpt from running the tool gosec:



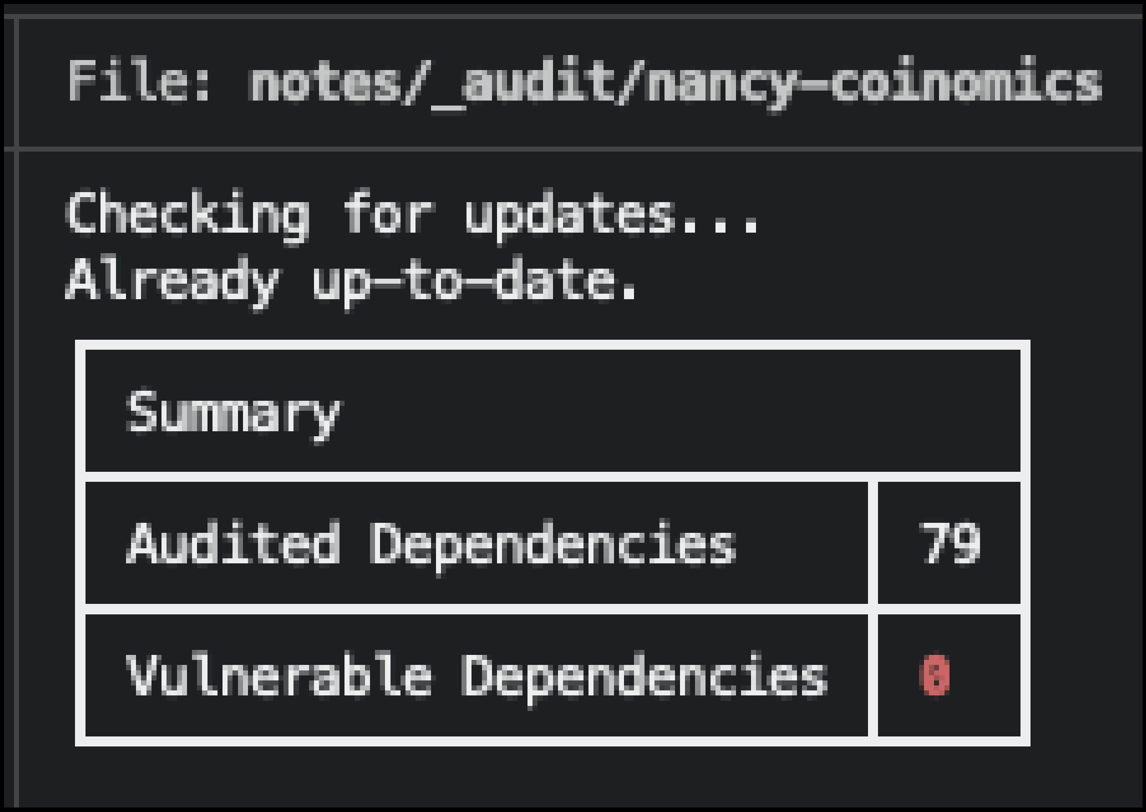
```
File: gosec
Results:


[/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 3: gosec excerpt

5.4 nancy

The tool nancy was used to search for known vulnerabilities within project dependencies. All dependencies were confirmed to contain no known vulnerabilities.

AUTOMATED TESTING



```
File: notes/_audit/nancy-coinomics

Checking for updates...
Already up-to-date.
┌─────────────────────────────────┐
│ Summary                         │
├────────────────────────────┬────┤
│ Audited Dependencies       │ 79 │
├────────────────────────────┼────┤
│ Vulnerable Dependencies    │ 0  │
└────────────────────────────┴────┘
```

Figure 4: Nancy excerpt

THANK YOU FOR CHOOSING

HALBORN