verichains

SECURITY AUDIT OF

ASTRA PROTOCOL

Public Report

Aug 04, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Aug 04, 2022. We would like to thank Astra for trusting Verichains Lab, delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the inflation module of the Astra Protocol. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the source code, along with some recommendations. Astra team has acknowledged and resolved these issues.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Astra Protocol

Astra is a scalable, high-throughput Proof-of-Stake blockchain that is fully compatible and interoperable with Ethereum. It's built using the Cosmos SDK which runs on top of Tendermint Core consensus engine.

Astra allows for running vanilla Ethereum as a Cosmos application-specific blockchain. This allows developers to have all the desired features of Ethereum, while at the same time, benefit from Tendermint's PoS implementation. Also, because it is built on top of the Cosmos SDK, it will be able to exchange value with the rest of the Cosmos Ecosystem through the Inter Blockchain Communication Protocol (IBC).

1.1.1. Inflation

In a Proof of Stake (PoS) blockchain, inflation is used as a tool to incentivize participation in the network. Inflation creates and distributes new tokens to participants who can use their tokens to either interact with the protocol or stake their assets to earn rewards and vote for governance proposals.

Especially in an early stage of a network, where staking rewards are high and there are fewer possibilities to interact with the network, inflation can be used as the major tool to incentivize staking and thereby securing the network.

With more stakers, the network becomes increasingly stable and decentralized. It becomes stable, because assets are locked up instead of causing price changes through trading. And it becomes decentralized, because the power to vote for governance proposals is distributed amongst more people.

1.1.2. Astra token model

The Astra Token Model outlines how the Astra network is secured through a balanced incentivized interest from users, developers and validators. In this model, inflation plays a major role in sustaining this balance. With an initial supply of 8,888,780,000 Astras and 111,111,000 Astras being issued through inflation during the first year, the model suggests an exponential decline in inflation to the rest of Astras in subsequent years.

Here are the detail of the tokenomics:

	%	#ASA	Minted at genesis	Vesting schedule
Core Contributors	15%	1,666,650,000	1,666,650,000	100% minted at genesis: 20% unlocked at genesis, 80% linearly vested in 8 years
Genesis Partners	15%	1,666,650,000	1,666,650,000	100% minted at genesis: 20% unlocked at genesis, 80% linearly vested in 8 years
Dev/Community Incentives	5%	555,550,000	555,550,000	100% minted at genesis: 20% unlocked at genesis, 80% linearly vested in 8 years
Staking Rewards	20%	2,222,220,000	0	100% exponentially vested with `decay_factor = 5%`
Reward Providers	30%	3,333,330,000	3,333,330,000	100% minted & locked at genesis, unlocked with GOV rules
Reserve Treasury	10%	1,111,110,000	1,111,110,000	100% minted & locked at genesis, unlocked with GOV rules
LP Rewards	5%	555,550,000	555,550,000	100% minted & locked at genesis, unlocked with GOV rules
Total	100%	11,111,000,000	8,888,780,000	

1.1.3. Exponential Inflation - Block Rewards

There will be a total of 20% of the total supply that will be distributed as block rewards which are mainly distributed as Staking Rewards. This module is dedicated to this allocation.

Block rewards are minted in `daily` epochs, via a decay function. During a period of 365 epochs (i.e, one year), a daily provision of Astra tokens is minted and allocated to staking rewards and reserve treasury. Within a period, the epoch provision does not change. The epoch provision is then reduced by a factor of `(1-r)` for subsequent years, with a decay factor `r`. Precisely, at the end of each period, the provision is recalculated as follows:

```
f(x) = r * (1 - r)^x * R
where
    x = variable = period (i.e, year)
    r = 0.1 = decay factor
    R = total amount of block rewards
```

With the given formula of `f(x)`, we can make sure that the total of minted block rewards never exceeds `R`, as:

$$\sum_{k=0}^{\infty} r(1-r)^k R = rR \sum_{k=0}^{\infty} (1-r)^k = \frac{rR}{1-(1-r)} = \frac{rR}{r} = R$$

With a decay factor of `0.05`, there will be a total of 40%, 64%, 87% and 98% of block rewards minted after the first 10, 20, 40, and 80 years, respectively. As a result, most of the block rewards will be distributed after 80 years. The decay factor can be changed via governance voting. A higher decay factor means it takes less time to mint most of the block rewards while a lower decay factor results in a longer minting period.

1.2. Audit scope

In this particular project, a timebox approach was used to define the consulting effort. This means that **Verichains Lab** allotted a prearranged amount of time to identify and document vulnerabilities. Because of this, there is no guarantee that the project has discovered all possible vulnerabilities and risks.

Furthermore, the security check is only an immediate evaluation of the situation at the time the check was performed. An evaluation of future security levels or possible future risks or vulnerabilities may not be derived from it.

This audit focused on identifying security flaws in code and the design of the inflation module of the Astra Protocol.

It was conducted on commit `d26568e5a8c999f1e6910fb85341ed92c6f06b53` from Git repository *https://github.com/AstraProtocol/astra*.

1.3. Audit methodology

Our security audit process includes three steps:

- Mechanism Design is reviewed to look for any potential problems.
- Source codes are scanned/tested for commonly known and more specific vulnerabilities using public and our in-house security analysis tool.
- Manual audit of the codes for security issues. The source code is manually analyzed to look for any potential problems.

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure application. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

Astra is built based on the Cosmos SDK, which is an open-source framework for building multi-asset public Proof-of-Stake (PoS) blockchains, like the Cosmos Hub, as well as permissioned Proof-of-Authority (PoA) blockchains. Blockchains built with the Cosmos SDK are generally referred to as application-specific blockchains. The default consensus engine available within the Cosmos SDK is Tendermint Core. Tendermint is the most (and only) mature BFT consensus engine in existence. It is widely used across the industry and is considered the gold standard consensus engine for building Proof-of-Stake systems.

Modules define most of the logic of Cosmos SDK applications. Developers compose modules together using the Cosmos SDK to build their custom application-specific blockchains. The inflation module is an important part of Astra, which control the logic of token minting and distributing for stakers.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities which were discovered by our team during the audit process. Astra team has updated the code, according to Verichains's draft report.

#	Issue	Severity	Status
1	Possibility of wrong calculation for `skippedEpochs`	LOW	Not fixed

Audit team also suggested some possible enhancements.

#	Issue	Status
1	The `periodProvision` formula is not well-documented	Acknowledged
2	Unused `BondedRatio` function	Fixed

2.2.1. Possibility of wrong calculation for `skippedEpochs` LOW

Affected files:

- x/inflation/keeper/hooks.go

In the `AfterEpochEnd` hook function, the `skippedEpochs` variable would be increased if the inflation is disabled and the current `epochIdentifier` is `"day"`. However, if the `epochIdentifier` is

configured to another value ("hour") as shown in the init.sh script, the value of skippedEpochs may be updated incorrectly.

```
# File: init.sh

cat $HOME/.astrad/config/genesis.json | jq
'.app_state["inflation"]["epoch_identifier"]="hour"' >
$HOME/.astrad/config/tmp_genesis.json && mv $HOME/.astrad/config/tmp_genesis.json
$HOME/.astrad/config/genesis.json
// File: x/inflation/keeper/hooks.go
func (k Keeper) AfterEpochEnd(ctx sdk.Context, epochIdentifier string, epochNumber int64) {
    params := k.GetParams(ctx)
    skippedEpochs := k.GetSkippedEpochs(ctx)

    // Skip inflation if it is disabled and increment number of skipped epochs
    if !params.EnableInflation {
        // check if the epochIdentifier is "day" before incrementing.
        if epochIdentifier != epochstypes.DayEpochID {
            return
        }
        skippedEpochs++ // INCORRECT IF epochIdentifier IS NOT "day"

        k.SetSkippedEpochs(ctx, skippedEpochs)
        k.Logger(ctx).Debug(
            "skipping inflation mint and distribution",
            "height", ctx.BlockHeight(),
            "epoch-id", epochIdentifier,
            "epoch-number", epochNumber,
            "skipped-epochs", skippedEpochs,
        )
        return
    }

    expEpochID := k.GetEpochIdentifier(ctx)
    if epochIdentifier != expEpochID {
        return
    }

    // mint coins, update supply
    epochMintProvision, found := k.GetEpochMintProvision(ctx)
    if !found {
        panic("the epochMintProvision was not found")
    }
    // ...
}
```

RECOMMENDATION

The value of `epochIdentifier` should always be checked with the `expEpochID` variable which is gotten from the store of inflation module as below:

```
expEpochID := k.GetEpochIdentifier(ctx)
```

UPDATES

- *Aug 01, 2022*: This issue won't be fixed since the `epochIdentifier` is always `day` on all platform. The issue will be migrated later when this identifier is changed since it might not be backward-compatible with the current codebase if the `epochIdentifier` is changed to something else other than `day`. The `hour` identifier in the `init.sh` file is for testing purposes only, and with `EnableInflation = true`, it will never take effect.

2.2.2. The `periodProvision` formula is not well-documented INFORMATIVE

Affected files:

- x/inflation/types/inflation_calculation.go

The current formula for `periodProvision` can be described as below:

```
periodProvision = exponentialDecay * MaxStakingRewards
f(x)            = r * (1 - r) ^ x * MaxStakingRewards

where (with default values):
x                 = variable    = year
r                 = 0.1         = decay factor
MaxStakingRewards = 2,222,200,000
```

Below is the corresponding function for calculating `periodProvision` and `epochProvision`.

```
// File: x/inflation/types/params.go
func DefaultParams() Params {
    return Params{
        MintDenom: DefaultInflationDenom,
        InflationParameters: InflationParameters{
            MaxStakingRewards:
sdk.NewDec(2222200000).Mul(ethermint.PowerReduction.ToDec()),
            R:                 sdk.NewDecWithPrec(10, 2), // decayFactor = 10%
        },
        EnableInflation: true,
    }
}

// File: x/inflation/types/inflation_calculation.go
func CalculateEpochMintProvision(
    params Params,
    period uint64,
    epochsPerPeriod int64,
) sdk.Dec {
```

```
    x := period                        // period
    r := params.InflationParameters.R // reduction factor

    // exponentialDecay := r * (1 - r) ^ x
    decay := sdk.OneDec().Sub(r)
    exponentialDecay := r.Mul(decay.Power(x))

    // periodProvision = exponentialDecay * maxStakingRewards
    periodProvision := exponentialDecay.Mul(params.InflationParameters.MaxStakingRewards)

    // epochProvision = periodProvision / epochsPerPeriod
    epochProvision := periodProvision.Quo(sdk.NewDec(epochsPerPeriod))

    // If the `denom` is already in `aastra`, multiply epochMintProvision with power
reduction (10^18 for astra)
    // as the issued tokens need to be given in `aastra`.
    if params.MintDenom == config.DisplayDenom {
        epochProvision = epochProvision.Mul(ethermint.PowerReduction.ToDec())
    }

    // The returned value is in `aastra`, therefore, it has to be rounded.
    return epochProvision.TruncateInt().ToDec()
}
```

The above function is quite important, but it is not well-documented. Please provide us with detailed explanations or documentation for this formula.

UPDATES

- *Aug 01, 2022*: Based on the provided documentation from Astra team, with the given formula `f(x) = r * (1 - r)^x * MaxStakingRewards`, we can make sure that the total minted block rewards never exceeds `MaxStakingRewards`. So, the severity of this issue has been changed from **MEDIUM** to **INFO**.

2.2.3. Unused `BondedRatio` function INFORMATIVE

Affected files:

- x/inflation/keeper/inflation.go

Since the `bondingIncentive` factor has been removed from the `periodProvision` formula, so the `BondedRatio` function is not needed anymore.

```
func (k Keeper) BondedRatio(ctx sdk.Context) sdk.Dec {
    stakeSupply := k.stakingKeeper.StakingTokenSupply(ctx)

    if !stakeSupply.IsPositive() {
        return sdk.ZeroDec()
    }
```

```
    return k.stakingKeeper.TotalBondedTokens(ctx).ToDec().QuoInt(stakeSupply)
}
```

UPDATES

- *Aug 01, 2022*: This issue has been acknowledged and fixed by removing the `BondedRatio` function.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 01, 2022*	Private Report	Verichains Lab
1.1	*Aug 03, 2022*	Public Report	Verichains Lab
1.2	*Aug 04, 2022*	Public Report	Verichains Lab

Table 2. Report versions history