Security Audit Report for Aura

Date: Mar 15, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2023.03.16 11:46:10
+08'00'

Contents

Report Manifest

Item	Description
Client	Aura Network
Target	Aura

Version History

Version	Date	Description
1.0	March 15, 2023	First Version

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Cosmos Chain
Language	Go
Approach	Semi-automatic and manual verification

The repository that has been audited includes aura-Aura_v0.4.3 [1].

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Aura	`Version 1`	`019eacad3805a0c5101904035bbbf13deed68b05`
	`Version 2`	`222d63e1aa5b6fee5aea689bcdfcb9af2dbc82a2`

Note that, we did **NOT** audit all the modules in the repository. The modules in this audit report covers **aura-Aura_v0.4.3** folder contract including five modules below:

- x/aura
- x/auth
- x/bank
- x/feegrant
- x/mint

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

[1] https://github.com/aura-nw/aura/releases/tag/Aura_v0.4.3

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Access control
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

∗ Duplicated item
∗ Verification of the token receiver
∗ Off-chain metadata security

1.3.4 Additional Recommendation

∗ Gas optimization
∗ Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find **four** potential issues. Besides, we have **two** recommendations and **three** notes as follows:

- High Risk: 0
- Medium Risk: 4
- Low Risk: 0
- Recommendations: 2
- Notes: 3

ID	Severity	Description	Category	Status
1	Medium	Lack of Check on parameter ExcludeCirculatingAddr	DeFi Security	Fixed
2	Medium	Incomplete Check in function ExcludeCirculatingAddr()	DeFi Security	Fixed
3	Medium	Incomplete Check in function CreatePeriodicVestingAccount()	DeFi Security	Fixed
4	Medium	No Limitation on Receiving Tokens for ExcludeCirculatingAddr	DeFi Security	Fixed
5	-	Insufficient Check of MaxSupply	Recommendation	Fixed
6	-	Gas Optimization	Recommendation	Fixed
7	-	Potential Effect on Minted Rewards	Note	Confirmed
8	-	Assumption on the Secure Implementation of Contract Dependencies	Note	Confirmed
9	-	Account Type of ExcludeCirculatingAddr	Note	Confirmed

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of Check on parameter ExcludeCirculatingAddr

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In module `aura`, the parameter `ExcludeCirculatingAddr` is not validated in the process of genesis state validation.

```go
// validate params
func (p Params) Validate() error {
    if err := validateMaxSupply(p.MaxSupply); err != nil {
        return err
    }

    return nil
}
```

Listing 2.1: aura/x/aura/types/params.go

Impact Illegal value of `ExcludeCirculatingAddr` can be initialized.

Suggestion Invoke the function `validateExcludeCirculatingAddr()` to check the state of the `ExcludeCirculatingAddr` in the function `Validate()`.

2.1.2 Incomplete Check in function ExcludeCirculatingAddr()

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The checks in the function `validateExcludeCirculatingAddr()` are incomplete. Specifically, the module needs to check each address is legitimate with fixed length and there are no repeated addresses in the `ExcludeCirculatingAddr`. Furthermore, the total length of the `ExcludeCirculatingAddr` is suggested to be checked with a reasonable threshold.

```go
func validateExcludeCirculatingAddr(i interface{}) error {
    v, ok := i.([]string)
    if !ok {
        return fmt.Errorf("invalid parameter type: %T", i)
    }

    for _, addBech32 := range v {
        if strings.TrimSpace(addBech32) == "" {
            return errors.New("exclude circulating address can not contain blank")
        }
    }
    return nil
}
```

Listing 2.2: aura/x/aura/types/params.go

Impact The number of coins held by `ExcludeCirculatingAddr` can influence the amount of minted rewards for each block. In this case, illegal `ExcludeCirculatingAddr` can result in unexpected block reward.

Suggestion Add the checks mentioned above in the function `validateExcludeCirculatingAddr()` accordingly.

2.1.3 Incomplete Check in function CreatePeriodicVestingAccount()

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In module `auth`, the user is able to create a periodic vesting account via the function `CreatePeriodicVestingAccount()`. It will transfer a certain amount of assets from the user's account to this newly created account. The assets will be released periodically according to the `StartTime` and `VestingPeriods` contained in the message.

However, the `StartTime` is unchecked, and it could be set ahead of the current block time, which is against the design of the function.

```go
func (s msgServer) CreatePeriodicVestingAccount(goCtx context.Context, msg *types.MsgCreatePeriodicVestingAccount) (*types.MsgCreatePeriodicVestingAccountResponse, error) {
    ctx := sdk.UnwrapSDKContext(goCtx)

    ak := s.AccountKeeper
    bk := s.BankKeeper

    from, err := sdk.AccAddressFromBech32(msg.FromAddress)
    if err != nil {
        return nil, err
    }
    to, err := sdk.AccAddressFromBech32(msg.ToAddress)
    if err != nil {
        return nil, err
    }

    if acc := ak.GetAccount(ctx, to); acc != nil {
        return nil, sdkerrors.Wrapf(sdkerrors.ErrInvalidRequest, "account %s already exists", msg.ToAddress)
    }

    var totalCoins sdk.Coins

    for _, period := range msg.VestingPeriods {
        totalCoins = totalCoins.Add(period.Amount...)
    }

    baseAccount := authtypes.NewBaseAccountWithAddress(to)
    baseAccount = ak.NewAccount(ctx, baseAccount).(*authtypes.BaseAccount)
    vestingAccount := org_types.NewPeriodicVestingAccount(baseAccount, totalCoins.Sort(), msg.StartTime, msg.VestingPeriods)

    ak.SetAccount(ctx, vestingAccount)

    defer func() {
        telemetry.IncrCounter(1, "new", "account")

        for _, a := range totalCoins {
            if a.Amount.IsInt64() {
                telemetry.SetGaugeWithLabels(
                    []string{"tx", "msg", "create_periodic_vesting_account"},
                    float32(a.Amount.Int64()),
                    []metrics.Label{telemetry.NewLabel("denom", a.Denom)},
                )
            }
        }
    }()

    err = bk.SendCoins(ctx, from, to, totalCoins)
    if err != nil {
        return nil, err
    }

    ctx.EventManager().EmitEvent(
        sdk.NewEvent(
            sdk.EventTypeMessage,
            sdk.NewAttribute(sdk.AttributeKeyModule, org_types.AttributeValueCategory),
        ),
    )
    return &types.MsgCreatePeriodicVestingAccountResponse{}, nil
}
```

Listing 2.3: aura/x/auth/vesting/msg_server.go

Impact Part of vesting assets will be released immediately. What's worse, if the `EndTime` is ahead of the current block time, all vesting assets will be released immediately.

Suggestion Add the check to make sure that the `StartTime` is larger than the current block time.

2.1.4 No Limitation on Receiving Tokens for ExcludeCirculatingAddr

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the current implementation, the assets held by `ExcludeCirculatingAddr` will not be counted into the staking total supply. The staking total supply will increase correspondingly if assets are transferred out to normal accounts.

According to the design, they will be created as vesting accounts whose assets will be locked at the beginning and released periodically. In this case, the staking total supply will increase periodically as expected. However, on the other side, if assets are transferred into these `ExcludeCirculatingAddr`, the staking total supply will decrease as well, which is insecure.

```
// BeginBlocker mints new tokens for the previous block.
func BeginBlocker(ctx sdk.Context, k custommint.Keeper) {
    defer telemetry.ModuleMeasureSince(types.ModuleName, time.Now(), telemetry.
        MetricKeyBeginBlocker)

    // fetch stored minter & params
    minter := k.GetMinter(ctx)
    params := k.GetParams(ctx)

    // check over max supply
    maxSupplyString := k.GetMaxSupply(ctx)
    maxSupply, ok := sdk.NewIntFromString(maxSupplyString)
    if !ok {
        panic(errors.New("panic convert max supply string to bigInt"))
    }
    k.Logger(ctx).Debug("Get max supply from aura", "maxSupply", maxSupply.String())
    currentSupply := k.GetSupply(ctx, params.GetMintDenom())
    k.Logger(ctx).Debug("Get current supply from network", "currentSupply", currentSupply.String()
        )

    excludeAmount := k.GetExcludeCirculatingAmount(ctx, params.GetMintDenom())
```

```go
	k.Logger(ctx).Debug("Exclude Addr", "exclude_addr", excludeAmount.String())

	if currentSupply.LT(maxSupply) {
		// recalculate inflation rate
		totalStakingSupply := k.CustomStakingTokenSupply(ctx, excludeAmount.Amount)
		bondedRatio := k.CustomBondedRatio(ctx, excludeAmount.Amount)
		k.Logger(ctx).Debug("Value BondedRatio: ", "bondedRatio", bondedRatio.String())
		minter.Inflation = minter.NextInflationRate(params, bondedRatio)
		minter.AnnualProvisions = minter.NextAnnualProvisions(params, totalStakingSupply)
		k.SetMinter(ctx, minter)

		// mint coins, update supply
		mintedCoin := minter.BlockProvision(params)
		mintedCoins := sdk.NewCoins(mintedCoin)

		supplyNext := currentSupply.Add(mintedCoin.Amount)
		if supplyNext.GT(maxSupply) {
			mintedCoin.Amount = maxSupply.Sub(currentSupply)
			mintedCoins = sdk.NewCoins(mintedCoin)
		}
		err := k.MintCoins(ctx, mintedCoins)
		if err != nil {
			panic(err)
		}

		// send the minted coins to the fee collector account
		err = k.AddCollectedFees(ctx, mintedCoins)
		if err != nil {
			panic(err)
		}

		if mintedCoin.Amount.IsInt64() {
			defer telemetry.ModuleSetGauge(types.ModuleName, float32(mintedCoin.Amount.Int64()), "
				minted_tokens")
		}

		ctx.EventManager().EmitEvent(
			sdk.NewEvent(
				types.EventTypeMint,
				sdk.NewAttribute(types.AttributeKeyBondedRatio, bondedRatio.String()),
				sdk.NewAttribute(types.AttributeKeyInflation, minter.Inflation.String()),
				sdk.NewAttribute(types.AttributeKeyAnnualProvisions, minter.AnnualProvisions.String
					()),
				sdk.NewAttribute(sdk.AttributeKeyAmount, mintedCoin.Amount.String()),
			),
		)

	} else {
		k.Logger(ctx).Info("Over the max supply", "currentSupply", currentSupply)
	}
}
```

Listing 2.4: aura/x/mint/abci.go

Impact The owner of ExcludeCirculatingAddr has the capability to manipulate the mint reward by transferring between ExcludeCirculatingAddr and normal accounts.

Suggestion Block ExcludeCirculatingAddr from receiving tokens.

2.2 Additional Recommendation

2.2.1 Insufficient Check of MaxSupply

Status Fixed in Version 2

Introduced by Version 1

Description In module aura, the check for parameter MaxSupply is insufficient. It only ensures the type of the parameter is correct, and no blank exists in it.

```go
func validateMaxSupply(i interface{}) error {
    v, ok := i.(string)
    if !ok {
        return fmt.Errorf("invalid parameter type: %T", i)
    }

    if strings.TrimSpace(v) == "" {
        return errors.New("max supply cannot be blank")
    }

    if !digitCheck.MatchString(strings.TrimSpace(v)) {
        return errors.New("invalid max supply parameter, expected string as number")
    }

    return nil
}
```

Listing 2.5: aura/x/aura/types/params.go

Suggestion I Add the check to make sure that the MaxSupply is equal or larger than a specified one.

2.2.2 Gas Optimization

Status Fixed in Version 2

Introduced by Version 1

Description Function CreatePeriodicVestingAccount() requires the provided address is not registered in module auth before. If it's already registered, the function will return an error.

However, the check uses the function GetAccount() in the account keeper to get the value of the account from KVStore, but never uses it in the following implementation, which is a waste of gas.

```go
func (s msgServer) CreatePeriodicVestingAccount(goCtx context.Context, msg *types.
    MsgCreatePeriodicVestingAccount) (*types.MsgCreatePeriodicVestingAccountResponse, error) {
  ctx := sdk.UnwrapSDKContext(goCtx)

  ak := s.AccountKeeper
  bk := s.BankKeeper
```

```go

from, err := sdk.AccAddressFromBech32(msg.FromAddress)
if err != nil {
  return nil, err
}
to, err := sdk.AccAddressFromBech32(msg.ToAddress)
if err != nil {
  return nil, err
}

if acc := ak.GetAccount(ctx, to); acc != nil {
  return nil, sdkerrors.Wrapf(sdkerrors.ErrInvalidRequest, "account %s already exists", msg.
      ToAddress)
}

var totalCoins sdk.Coins

for _, period := range msg.VestingPeriods {
  totalCoins = totalCoins.Add(period.Amount...)
}

baseAccount := authtypes.NewBaseAccountWithAddress(to)
baseAccount = ak.NewAccount(ctx, baseAccount).(*authtypes.BaseAccount)
vestingAccount := org_types.NewPeriodicVestingAccount(baseAccount, totalCoins.Sort(), msg.
    StartTime, msg.VestingPeriods)

ak.SetAccount(ctx, vestingAccount)

defer func() {
  telemetry.IncrCounter(1, "new", "account")

  for _, a := range totalCoins {
    if a.Amount.IsInt64() {
      telemetry.SetGaugeWithLabels(
        []string{"tx", "msg", "create_periodic_vesting_account"},
        float32(a.Amount.Int64()),
        []metrics.Label{telemetry.NewLabel("denom", a.Denom)},
      )
    }
  }
}()

err = bk.SendCoins(ctx, from, to, totalCoins)
if err != nil {
  return nil, err
}

ctx.EventManager().EmitEvent(
  sdk.NewEvent(
  sdk.EventTypeMessage,
    sdk.NewAttribute(sdk.AttributeKeyModule, org_types.AttributeValueCategory),
  ),
)
```

```
    return &types.MsgCreatePeriodicVestingAccountResponse{}, nil
}
```

Listing 2.6: aura/x/auth/vesting/msg_server.go

Suggestion I It's suggested to replace the function `GetAccount()` with the function `HasAccount()`.

2.3 Notes

2.3.1 Potential Effect on Minted Rewards

Status Confirmed

Introduced by `version 1`

Description The `Aura` chain introduces a few privileged addresses whose assets will not be counted into the custom staking token total supply. Since the amount of minted rewards for each block is calculated based on the total supply, the balance change of these addresses may affect the minted rewards.

2.3.2 Assumption on the Secure Implementation of Contract Dependencies

Status Confirmed

Introduced by `version 1`

Description The `Aura` chain is mainly built based on `Wasmd` (version 0.29.1), `Cosmos-SDK` (version 0.45.14) and `IBC-Go` (version 3.3.0). In this audit, we assume the implementation provided by the standard library has no security issues.

2.3.3 Account Type of ExcludeCirculatingAddr

Status Confirmed

Introduced by `version 1`

Description According to the design of `Aura`, `ExcludeCirculatingAddr` will be created as vesting accounts in the genesis block so that their locked assets will be released periodically to increase the total supply.