HAECHI AUDIT

MyTVchain

Smart Contract Security Analysis

Published on : Nov 23, 2021

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

MyTVchain

Security Report Published by HAECHI AUDIT
v1.0 Nov 10, 2021
v2.0 Nov 23, 2021

Auditor : Hoon Won

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	1	1	-	-	-
Minor	4	3	-	-	-
Tips	1	1	-	-	-

TABLE OF CONTENTS

5 Issues (0 Critical, 1 Major, 4 Minor) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring. We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of MyTV smart contract created by MyTVchain team. HAECHI AUDIT focused on whether the smart contract created by MyTVchain team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends MyTVchain team improve all issues discovered. The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, while *Sample#fallback()* means the fallback() function of the Sample contract. Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/smart-chain-fr/smartcontractMyTV/tree/main/packages/hardhat/contracts). The last commit of the code used in this Audit is "01f6569a606026572a673e7146e50ee7f06523d6".

Issues HAECHI AUDIT found 0 critical issues, 1 major issue, and 4 minor issues. There is 1 Tips issue that can improve the code's usability or efficiency upon modification.

update [v.2.0] In the new commit b394a3a2026a1d0d194cd9eaa87d9b69f959b9ee, 1 major issue, 3 minor issues, and 1 Tips issue have been revised.

Severity	Issue	Status
⚠ MAJOR	The id information of MyTvStaking#stakingFlexKeys may mismatch with the id information of MyTvStaking#stakingFlex.	(Found - v1.0) (Resolved - v2.0)
MINOR	In the MyTvStaking#unstakeLock() function, MyTvStaking#penaltyBalance value may be updated as unintended.	(Found - v1.0) (Resolved - v2.0)
MINOR	StakingPack already in progress can call the MyTvStaking#deleteStakingPack() function.	(Found - v1.0) (Resolved - v2.0)
MINOR	The reward value of StakingFlex is not 0 in the MyTvStaking#stakeFor() function.	(Found - v1.0) (Resolved - v2.0)
MINOR	Even when MyTvStaking#unstakeFlex() is called to unstake all the amount, the currentUser information is not updated.	(Found - v1.0)

💡 **TIPS**	There are missing Events.	(Found - v1.0) (Resolved - v2.0)

OVERVIEW

Contracts subject to audit

- ❖ MyTvGovernanceToken
- ❖ MyTvLock
- ❖ MyTvStaking
- ❖ MyTvFarming

FINDINGS

⚠️ **MAJOR**

The id information of MyTvStaking#stakingFlexKeys may mismatch with the id information of MyTvStaking#stakingFlex.

(Found - v.1.0) (Resolved - v.2.0)

```
530    function unstakeFlex(uint256 amount) external nonReentrant returns (bool) {
531        StakingFlex storage stakingFlex = stakedFlex[msg.sender];
532        require(stakingFlex.amount ≥ amount, "Amount exceeds staked balance");
533        require(amount > 0, "Unstake amount cannot be 0");
534        uint256 reward = stakingFlex.reward.add(
535            getReward(
536                stakingFlex.timestamp,
537                block.timestamp,
538                stakingFlex.amount,
539                stakingFlex.rate
540            )
541        );
542        require(
543            myTvGovernanceToken.balanceOf(address(this)) > amount.add(reward),
544            "Staking Contract cannot pay rewards"
545        );
546        uint256 stakedBalance = stakingFlex.amount;
547        stakingFlex.amount = stakingFlex.amount.sub(amount);
548        stakingFlex.timestamp = block.timestamp;
549        balancesStaked[msg.sender] = balancesStaked[msg.sender].sub(amount);
550        totalStake = totalStake.sub(amount);
551
552        if (amount == stakedBalance) {
553            stakedFlexKeys[stakingFlex.id] = stakedFlexKeys[
554                stakedFlexKeys.length.sub(1)
555            ];
556            stakedFlexKeys.pop();
557        }
558        myTvGovernanceToken.safeTransfer(msg.sender, amount.add(reward));
559        emit Unstake(msg.sender, 0, amount.add(reward));
560        return true;
561    }
```

[https://github.com/smart-chain-fr/smartcontractMyTV/blob/main/packages/hardhat/contracts/MyTvStaking.sol#L530-L561]

Issue

When a user unstakes all the amount staked in flexible stakePack, the *MyTvStaking#unstakeFlex()* function removes the user's address from *MyTvStaking#stakedFlexKeys* array. Because the removal first swaps the last element of the array with the user then removes the last element, another user's index (id) corresponding to the last element is changed.

However, at this time, another user's *MyTvStaking#stakedFlex* id info is not updated together, causing a mismatch of the id info between *MyTvStaking#stakedFlexKeys* and *MyTvStaking#stakedFlex*.

Recommendation

We recommend adding a logic that also modifies another user's *MyTvStaking#stakedFlex* id information when modifying the *MyTvStaking#stakedFlexKeys* array in the *MyTvStaking#unstakeFlex()* function.

Update

[v2.0] - The issue has been resolved by removing the logic in which the *MyTvStaking#unstakeFlex()* function removes the user's address from the *MyTvStaking#stakedFlexKeys* array when the user unstakes all staked amounts in the flexible stakePack.

MINOR

In the MyTvStaking#unstakeLock() function, MyTvStaking#penaltyBalance value may be updated as unintended.

(Found - v.1.0) (Resolved - v.2.0)

```
function unstakeLock(uint256 stakeId)
    external
    nonReentrant
    returns (uint256)
{
   uint256 amountUnstaked;
    uint256 penaltyBalance_ = 0;
    require(
        stakeId < staked[msg.sender].length,
        "User stake does not exist"
    );
    Staking storage staking = staked[msg.sender][stakeId];
    uint256 packId_ = staking.packId;
    if (block.timestamp < (staking.timestamp).add(staking.period)) {
        require(staking.unlockable, "Can't unlock this staking");
        uint256 penalty = getReward(
            block.timestamp,
            block.timestamp.add(365 days),
            staking.amount,
            staking.feesOnStakeIfUnstakeEarlier
        );
        amountUnstaked = staking.amount.sub(penalty);
        penaltyBalance_ = penaltyBalance.add(
            staking.reward.sub(staking.claimed)
        );
    } else {
        amountUnstaked = staking.amount.add(
            (staking.reward).sub(staking.claimed)
        );
    }
    require(
        myTvGovernanceToken.balanceOf(address(this)) > amountUnstaked,
        "Staking Contract cannot pay"
    );
    penaltyBalance = penaltyBalance_;
    balancesStaked[msg.sender] = balancesStaked[msg.sender].sub(
        staking.amount
    );
    totalStake = totalStake.sub(staking.amount);
```

```
680         stakingOptions[staking.packId].currentUser = stakingOptions[
681             staking.packId
682         ]
683         .currentUser
684         .sub(1);
685         staked[msg.sender][stakeId] = staked[msg.sender][
686             staked[msg.sender].length.sub(1)
687        ];
688         staked[msg.sender].pop();
689
690         myTvGovernanceToken.safeTransfer(msg.sender, amountUnstaked);
691         emit Unstake(msg.sender, packId_, amountUnstaked);
692         return amountUnstaked;
693     }
```

[https://github.com/smart-chain-fr/smartcontractMyTV/blob/main/packages/hardhat/contracts/MyTvStaking.sol#L641-L693]

Issue

MyTvStaking#penaltyBalance is identified as a variable that sets a penalty of a certain percentage when a user unstakes or claims earlier than the period of the stacking pack, and that accumulates and stores thereof. However, in the case of *MyTvStaking#L663-665* and *MyTvStaking#L675*, when a user calls the *MyTvStaking#unstakeLock()* function earlier than the set period, they add and update an unclaimed reward, not the penalty, to *MyTvStaking#penaltyBalance* . This likely conflicts with the intention to accumulate and store penalties in *MyTvStaking#penaltyBalance*.

Furthermore, when a user calls the *MyTvStaking#unstakeLock()* function after the set period has elapsed, *MyTvStaking#penaltyBalance* is updated to penaltyBalance_, which was initialized to 0, by *MyTvStaking#L675*, causing the value stored earlier to disappear. This also likely conflicts with the intention.

Recommendation

We recommend modifying the logic appropriately so that *MyTvStaking#penaltyBalance* updates can occur correctly.

Update

[v2.0] - The issue has been resolved by modifying the logic so that *MyTvStaking#penaltyBalance* can be properly accumulated.

MINOR

StakingPack already in progress can call the MyTvStaking#deleteStakingPack() function.

(Found - v.1.0) (Resolved - v.2.0)

```
317    function deleteStakingPack(uint256 packId)
318        external
319        onlyAllowed
320        packIdExists(packId)
321        returns (bool)
322    {
323        require(packId != 0, "Cannot delete flex pack");
324        stakingOptions[packId] = stakingOptions[stakingOptions.length - 1];
325        stakingOptions.pop();
326        emit StakingPackDeleted(packId);
327        return true;
328    }
```

[https://github.com/smart-chain-fr/smartcontractMyTV/blob/main/packages/hardhat/contracts/MyTvStaking.sol#L317-L328]

```
641    function unstakeLock(uint256 stakeId)
642        external
643        nonReentrant
644        returns (uint256)
645    {
646       uint256 amountUnstaked;
647        uint256 penaltyBalance_ = 0;
648        require(
649            stakeId < staked[msg.sender].length,
650            "User stake does not exist"
651        );
652        Staking storage staking = staked[msg.sender][stakeId];
653        uint256 packId_ = staking.packId;
654        if (block.timestamp < (staking.timestamp).add(staking.period)) {
655            require(staking.unlockable, "Can't unlock this staking");
656            uint256 penalty = getReward(
657                block.timestamp,
658                block.timestamp.add(365 days),
659                staking.amount,
660                staking.feesOnStakeIfUnstakeEarlier
661            );
662            amountUnstaked = staking.amount.sub(penalty);
663            penaltyBalance_ = penaltyBalance.add(
664                staking.reward.sub(staking.claimed)
665            );
```

```
666         } else {
667             amountUnstaked = staking.amount.add(
668                 (staking.reward).sub(staking.claimed)
669             );
670         }
671         require(
672             myTvGovernanceToken.balanceOf(address(this)) > amountUnstaked,
673             "Staking Contract cannot pay"
674         );
675         penaltyBalance = penaltyBalance_;
676         balancesStaked[msg.sender] = balancesStaked[msg.sender].sub(
677             staking.amount
678         );
679         totalStake = totalStake.sub(staking.amount);
680         stakingOptions[staking.packId].currentUser = stakingOptions[
681             staking.packId
682         ]
683         .currentUser
684         .sub(1);
685         staked[msg.sender][stakeId] = staked[msg.sender][
686             staked[msg.sender].length.sub(1)
687        ];
688         staked[msg.sender].pop();
689
690         myTvGovernanceToken.safeTransfer(msg.sender, amountUnstaked);
691         emit Unstake(msg.sender, packId_, amountUnstaked);
692         return amountUnstaked;
693     }
```

[https://github.com/smart-chain-fr/smartcontractMyTV/blob/main/packages/hardhat/contracts/MyTvStaking.sol#L641-L693]

Issue

The *MyTvStaking#deleteStakingPack()* function removes the index element corresponding to the packId received as a parameter from the *MyTvStaking#stakingOptions* array. However, when there is a user who has already participated in the removed StakingPack, there may be a problem in *MyTvStaking#L680* trying to access the non-existent element of the *MyTvStaking#stakingOptions* array when the user calls the *MyTvStaking#unstakeLock()* function afterward.

Recommendation

We recommend adding a require() statement that requires there is no ongoing StakingPack corresponding to the packId received as a parameter in the *MyTvStaking#deleteStakingPack()* function.

Acknowledgement

If the implementation was intended, no modification is necessary.

Update

[v2.0] - The issue has been resolved by deleting the *MyTvStaking#deleteStakingPack()* function.

MINOR

The reward value of StakingFlex is not 0 in the MyTvStaking#stakeFor() function.

(Found - v.1.0) (Resolved - v.2.0)

```
function stakeFor(
    address from,
    uint256 amount,
    uint256 packId
) external onlyOwner packIdExists(packId) returns (bool) {
    require(stakeForEnabled, "This function is disabled");
    StakingPack storage stakingPack = stakingOptions[packId];
    require(amount > stakingPack.minStake, "Amount < minStake");
    require(
        amount < stakingPack.maxStake || stakingPack.maxStake == 0,
        "Amount > maxStake"
    );
    require(
        stakingPack.currentUser < stakingPack.maxUser ||
            stakingPack.maxUser == 0,
        "This pack is not available"
    );

   uint256 reward = getReward(
        block.timestamp,
        block.timestamp.add(stakingPack.period),
        amount,
        stakingPack.rate
    );

    require(
        myTvGovernanceToken.balanceOf(address(this)) > amount.add(reward),
        "Staking Contract cannot allocate stake"
    );

    if (packId > 0) {
        staked[from].push(
            Staking(
                stakingPack.period,
                stakingPack.rate,
                block.timestamp,
                amount,
                reward,
                stakingPack.unlockable,
                stakingPack.claimable,
```

```
740                     0,
741                     stakingPack.feesOnRewardIfUnstakeEarlier,
742                     stakingPack.feesOnStakeIfUnstakeEarlier,
743                     packId
744                 )
745             );
746         } else {
747             require(
748                 stakedFlex[from].amount == 0,
749                 "You already have a staking flex"
750             );
751             stakedFlex[from] = StakingFlex(
752                 stakingPack.rate,
753                 block.timestamp,
754                 amount,
755                 reward,
756                 stakedFlexKeys.length
757             );
758             stakedFlexKeys.push(from);
759         }
760         stakingPack.currentUser = stakingPack.currentUser.add(1);
761         balancesStaked[from] = balancesStaked[from].add(amount);
762         totalStake = totalStake.add(amount);
763 
764         emit Stake(from, packId, amount, stakingPack.period, stakingPack.rate);
765         return true;
766     }
```

[https://github.com/smart-chain-fr/smartcontractMyTV/blob/main/packages/hardhat/contracts/MyTvStaking.sol#L700-L766]

Issue

It is identified that Flexible staking pack acts as intended when the reward value is entered as 0 in *MyTvStaking#stakedFlex* when staking. However, in the case of the *MyTvStaking#stakeFor()*function, the reward value enters *MyTvStaking#stakedFlex* as a significant value, unlike staking in the flexible stacking pack with the *MyTvStaking#stake()* function. This can cause unintended behavior in the *MyTvStaking#increaseStakeFlex()* function and other places.

Recommendation

We advise to input 0 as the reward value in *MyTvStaking#stakedFlex* if packId is 0 when the *MyTvStaking#stakeFor()* function is called.

Acknowledgment

If the implementation was intended, no modification is necessary.

Update

[v2.0] - The issue has been resolved by removing the reward attribute from the *MyTvStaking#stakedFlex* struct and eliminating the logic that uses the reward in *MyTvStaking#stakedFlex* from the *MyTvStaking#increaseStakeFlex()* function, etc.

MINOR

Even when MyTvStaking#unstakeFlex() is called to unstake all the amount, the currentUser information is not updated.

(Found - v.1.0)

```
530    function unstakeFlex(uint256 amount) external nonReentrant returns (bool) {
531        StakingFlex storage stakingFlex = stakedFlex[msg.sender];
532        require(stakingFlex.amount ≥ amount, "Amount exceeds staked balance");
533        require(amount > 0, "Unstake amount cannot be 0");
534        uint256 reward = stakingFlex.reward.add(
535            getReward(
536                stakingFlex.timestamp,
537                block.timestamp,
538                stakingFlex.amount,
539                stakingFlex.rate
540            )
541        );
542        require(
543            myTvGovernanceToken.balanceOf(address(this)) > amount.add(reward),
544            "Staking Contract cannot pay rewards"
545        );
546        uint256 stakedBalance = stakingFlex.amount;
547        stakingFlex.amount = stakingFlex.amount.sub(amount);
548        stakingFlex.timestamp = block.timestamp;
549        balancesStaked[msg.sender] = balancesStaked[msg.sender].sub(amount);
550        totalStake = totalStake.sub(amount);
551
552        if (amount == stakedBalance) {
553            stakedFlexKeys[stakingFlex.id] = stakedFlexKeys[
554                stakedFlexKeys.length.sub(1)
555            ];
556            stakedFlexKeys.pop();
557        }
558        myTvGovernanceToken.safeTransfer(msg.sender, amount.add(reward));
559        emit Unstake(msg.sender, 0, amount.add(reward));
560        return true;
561    }
```

[https://github.com/smart-chain-fr/smartcontractMyTV/blob/main/packages/hardhat/contracts/MyTvStaking.sol#L530-L561]

Issue

When a user unstakes all staked amount in the flexible staking pack, the *MyTvStaking#unstakeFlex()* function removes the user's address from the *MyTvStaking#stakedFlexKeys* array. However, inside the function, it does not update the currentUser of the corresponding stake pack. This is likely to be an unintended behavior.

Recommendation

We recommend adding a statement that updates the currentUSer of the stake pack inside the function when a user calls the *MyTvStaking#unstakeFlex()* function to unstake all the staked amount.

Acknowledgement

If the implementation was intended, no modification is necessary.

💡 **TIPS**

There are missing events.

(Found - v.1.0) (Resolved - v.2.0)

The following is a list of functions with missing Events.

Function	Expected Event	Emitted Event	Omitted Event
transferToReserveAndBurn	TransferReserve, Burn	Burn	TransferReserve

Without Event, it is difficult to identify in real-time whether accurate values are recorded on the blockchain. In this case, it becomes problematic to determine whether the corresponding value has been changed in the application and whether the corresponding function has been called.

Thus, we recommended adding Events corresponding to the change occurring in the function.

Update

[v2.0] - Events have been appropriately added.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

```
SafeERC20
  with address that has no contract code
    ✓ reverts on transfer
    ✓ reverts on transferFrom
    ✓ reverts on approve
    ✓ reverts on increaseAllowance
    ✓ reverts on decreaseAllowance
  with token that returns false on all calls
    ✓ reverts on transfer
    ✓ reverts on transferFrom
    ✓ reverts on approve
    ✓ reverts on increaseAllowance
    ✓ reverts on decreaseAllowance
  with token that returns true on all calls
    ✓ doesn't revert on transfer
    ✓ doesn't revert on transferFrom
    approvals
      with zero allowance
        ✓ doesn't revert when approving a non-zero allowance
        ✓ doesn't revert when approving a zero allowance
        ✓ doesn't revert when increasing the allowance
        ✓ reverts when decreasing the allowance
      with non-zero allowance
        ✓ reverts when approving a non-zero allowance
        ✓ doesn't revert when approving a zero allowance
        ✓ doesn't revert when increasing the allowance
        ✓ doesn't revert when decreasing the allowance to a positive value
        ✓ reverts when decreasing the allowance to a negative value
  with token that returns no boolean values
    ✓ doesn't revert on transfer
    ✓ doesn't revert on transferFrom
    approvals
      with zero allowance
        ✓ doesn't revert when approving a non-zero allowance
        ✓ doesn't revert when approving a zero allowance
```

```
      ✓ doesn't revert when increasing the allowance
      ✓ reverts when decreasing the allowance
    with non-zero allowance
      ✓ reverts when approving a non-zero allowance
      ✓ doesn't revert when approving a zero allowance
      ✓ doesn't revert when increasing the allowance
      ✓ doesn't revert when decreasing the allowance to a positive value
      ✓ reverts when decreasing the allowance to a negative value

SafeMath
  add
    ✓ adds correctly
    ✓ reverts on addition overflow
  sub
    ✓ subtracts correctly
    ✓ reverts if subtraction result would be negative
  mul
    ✓ multiplies correctly
    ✓ multiplies by zero correctly
    ✓ reverts on multiplication overflow
  div
    ✓ divides correctly
    ✓ divides zero correctly
    ✓ returns complete number result on non-even division
    ✓ reverts on division by zero
  mod
    ✓ reverts with a 0 divisor
    modulos correctly
      ✓ when the dividend is smaller than the divisor
      ✓ when the dividend is equal to the divisor
      ✓ when the dividend is larger than the divisor
      ✓ when the dividend is a multiple of the divisor

ERC20Snapshot
  #_snapshot()
    ✓ should creates increasing snapshots ids, starting from 1
    valid case
      ✓ should emits a snapshot event
  #totalSupplyAt()
    ✓ should fail if snapshot id is zero
    ✓ should fail if snapshot is not-yet-created
    valid case
      with no supply changes after the snapshot
        ✓ should return current total supply
```

```
    with supply changes after the snapshot
      ✓ should return total supply before the changes
      with a second snapshot after supply changes
        ✓ should return the supply before and after the changes
      with multiple snapshots after supply changes
        ✓ all posterior snapshots should return supply after the changes
  #balanceOfAt()
    ✓ should fail if snapshot id is zero
    ✓ should fail if snapshout is not-yet-created
    valid case
      with no balance changes after the snapshot
        ✓ should return current balance for all accounts
      with balance changes after the snapshot
        ✓ should returns the balances before the changes
      with a second snapshot after supply changes
        ✓ snapshots should return the balances before and after the changes
      with multiple snapshots after supply changes
        ✓ all posterior snapshots should return the supply after the changes (69ms)

MyTvFarming
  #add()
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ new pool should be pushed to poolInfo
      ✓ totalAllocPoint should increase
      ✓ should emit Add event
  #set()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if pool id does not exist
    valid case
      ✓ pool should be updated properly
      ✓ totalAllocPoint should be updated properly
  #updatePool()
    ✓ should fail if pool id does not exist
    ✓ should not update if block.number is less than lastRewardBlock
    ✓ should not update except lastRewardBlock if lpSupply is zero
    ✓ should fail if staking contract does not have enough funds to pay rewards
    valid case
      ✓ should update pool properly
  #deposit()
    ✓ should fail if pool id does not exist
    ✓ should fail if msg.sender's balance is less than amount
    ✓ should fail if msg.sender's allowance for lpToken contract is less than amount
    valid case
```

```
    ✓ should deposit in pool
    ✓ lpToken contract's balance should increase
    ✓ msg.sender's balance should decresase
    ✓ should emit Deposit event (74ms)
  #withdraw()
    ✓ should fail if pool id does not exist
    ✓ should fail if withdraw amount is greater than deposit amount
    ✓ should fail if staking contract does not have enough funds to pay rewards (40ms)
    valid case1: partial withdraw
      ✓ should withdraw properly (74ms)
      ✓ should emit Withdraw event (59ms)
    valid case2: withdraw all
      ✓ should withdraw properly (64ms)
      ✓ should emit Withdraw event (58ms)

MyTvGovernanceToken
  #constructor()
    ✓ should set name properly
    ✓ should set symbol properly
    ✓ should set decimals properly
    ✓ should set initial supply properly
  ERC20 Spec
    #transfer()
      ✓ should fail if recipient is ZERO_ADDRESS
      ✓ should fail if sender's amount is lower than balance
      when succeeded
        ✓ sender's balance should decrease
        ✓ recipient's balance should increase
        ✓ should emit Transfer event
    #transferFrom()
      ✓ should fail if sender is ZERO_ADDRESS
      ✓ should fail if recipient is ZERO_ADDRESS
      ✓ should fail if sender's amount is lower than transfer amount
      ✓ should fail if allowance is lower than transfer amount
      ✓ should fail even if try to transfer sender's token without approve process
      when succeeded
        ✓ sender's balance should decrease
        ✓ recipient's balance should increase
        ✓ should emit Transfer event
        ✓ allowance should decrease
        ✓ should emit Approval event
    #approve()
      ✓ should fail if spender is ZERO_ADDRESS
      valid case
```

```
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #increaseAllowance()
    ✓ should fail if spender is ZERO_ADDRESS
    ✓ should fail if overflows
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #decreaseAllowance()
    ✓ should fail if spender is ZERO_ADDRESS
    ✓ should fail if overflows
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
ERC20 Burnable Spec
  #burn()
    ✓ should fail if try to burn more than burner's balance
    valid case
      ✓ totalSupply should decrease
      ✓ account's balance should decrease
      ✓ should emit Transfer event
      ✓ should emit Burn event
MyTvGovernanceToken Spec
  #setStakingAddress()
    ✓ should fail if msg.sender is not owner nor votingContract
    ✓ should fail if msg.sender is AddressZero
    valid case
      ✓ stakingAddress should be set properly
      ✓ should emit StakingAddressUpdated Event
  #setVotingContract()
    ✓ should fail if msg.sender is not owner nor votingContract
    ✓ should fail if msg.sender is AddressZero
    valid case
      ✓ votingContract should be set properly
      ✓ should emit VotingContractUpdated Event
  #updateBurnPercentage()
    ✓ should fail if msg.sender is not owner nor votingContract
    valid case
      ✓ votingContract should be set properly
      ✓ should emit BurnPercentageUpdated Event
  #transferToReserve()
    ✓ should fail if msg.sender's balance is less than amount
    ✓ should fail if the stakingAddress has not yet been set
    ✓ should fail if amount is zero
```

```
    valid case
      ✓ stakingAddress' balance should increase
      ✓ msg.sender's balance should decrease
      ✓ should emit TransferReserve Event
  #transferToReserveAndBurn()
    ✓ should fail if msg.sender's balance is less than amount
    ✓ should fail if the stakingAddress has not yet been set
    ✓ should fail if amount is zero
    valid case
      ✓ stakingAddress' balance should increase
      ✓ msg.sender's balance should decrease
      1) should emit TransferReserve Event
      2) should emit burn Event
  #snapshot()
    ✓ should fail if msg.sender is not owner
    ✓ should creates increasing snapshots ids, starting from 1
    valid case
      ✓ should emits a snapshot event

MyTvLock
  #constructor()
    ✓ myTvGovernanceToken should be set properly
    ✓ lockForEnabled should be true
  #disableLockFor()
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ lockForEnabled should be false
      ✓ should emit DisableLockForUpdated event
  #lockFor()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if lockFor function is disabled
    ✓ should fail if contract's unlocked balance is not greater than amount
    valid case
      ✓ AddressToTokenLock info for from address should be set properly
      ✓ totalLocked should increase
      ✓ should emit TokenLocked event
  #unlockToken()
    ✓ should fail if token lock info for msg.sender does not exist
    ✓ should fail if msg.sender already unlocked all token
    ✓ should fail if less than 30 days have passed since the token was locked
    ✓ should fail if contract's balance is less than unlocked amount
    valid case
      after 30days
        ✓ msg.sender's balance should not change
```

```
      ✓ contract's balance should not change
      ✓ totalLocked should not change
      ✓ claimed should not change
      ✓ should emit TokenUnlocked event
    after (30 + 50)days
      ✓ msg.sender's balance should increase
      ✓ contract's balance should decrease
      ✓ totalLocked should decrease
      ✓ claimed should increase
      ✓ should emit TokenUnlocked event
    after (30 + 50 + 60)days
      ✓ msg.sender's balance should increase
      ✓ contract's balance should decrease
      ✓ totalLocked should decrease
      ✓ claimed should increase
      ✓ should emit TokenUnlocked event
    after (30 + 50 + 60 + 100)days
      ✓ should fail if call #unlockToken() function

MyTvStaking
  #constructor()
    ✓ myTvRewardAddress should be AddressZero
    ✓ myTvRewardAmount should be zero
    ✓ myTvRewardPeriod should be zero
    ✓ totalStake should be zero
    ✓ stakeForEnabled should be true
    ✓ myTvGovernanceToken should be set properly
  #disableStakeFor()
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ lockForEnabled should be false
      ✓ should emit StakeForEnabledUpdated event
  #updateMyTvRewardAmount()
    ✓ should fail if msg.sender is not owner nor votingContract
    valid case
      ✓ myTvRewardAmount should be set properly
      ✓ should emit MyTvRewardAmountUpdated Event
  #updateMyTvRewardAddress()
    ✓ should fail if msg.sender is not owner nor votingContract
    ✓ should fail if newAddress is AddressZero
    valid case
      ✓ myTvRewardAddress should be set properly
      ✓ should emit MyTvRewardAddressUpdated Event
  #updateMyTvRewardPeriod()
```

```
  ✓ should fail if msg.sender is not owner nor votingContract
  ✓ should fail if newPeriod is zero
  valid case
    ✓ myTvRewardAddress should be set properly
    ✓ should emit MyTvRewardPeriodUpdated Event
#addStakingPack()
  ✓ should fail if msg.sender is not owner nor votingContract
  ✓ should fail if minStake is greater than or equal maxStake
  valid case
    ✓ stakingPack info should be pushed to stakingOptions properly
    ✓ should emit StakingPackAdded event
#setStakingPack()
  ✓ should fail if msg.sender is not owner nor votingContract
  ✓ should fail if the stakingPack corresponding to packId does not exist
  ✓ should fail if minStake is greater than or equal maxStake
  valid case
    ✓ stakingPack info should be updated properly
    ✓ should emit StakingPackUpdated event
#deleteStakingPack()
  ✓ should fail if msg.sender is not owner nor votingContract
  ✓ should fail if packId does not exist
  ✓ should fail if packId is zero
  3) should fail if the stakeOption of packId is in progress
  valid case
    ✓ stakingPack info should be deleted properly (62ms)
    ✓ should emit StakingPackUpdated event
#stake()
  ✓ should fail if the stakingPack corresponding to packId does not exist,
  ✓ should fail if onlyAdmin is true
  ✓ should fail if amount is less than minStake
  ✓ should fail if maxStake is not zero and amount is greater than maxStake
  ✓ should fail if currentUser is equal to maxUser (96ms)
  ✓ should fail if contract's balance is less than reward
  ✓ should fail if msg.sender already have a staking flex and try to flexible stake (45ms)
  ✓ should fail if msg.sender's balance is less than amount (40ms)
  ✓ should fail if msg.sender's allowance for contract is less than amount
  valid case
    ✓ contract's balance should increase
    ✓ msg.sender's balance should decrease
    ✓ currentUser should increase
    ✓ balancesStaked should increase
    ✓ totalStaked should increase
    ✓ should emit Stake event
    case1: non flexible staking
```

```
      ✓ staking info pushed to msg.sender's staked info array properly
    case2: flexible staking
      ✓ msg.sender's stakedFlex info should be set properly
      ✓ msg.sender's address should be pushed to stakedFlexKeys array
  #updateStakeFlexRate()
    ✓ should fail if msg.sender is not owner (87ms)
    ✓ should fail if x is greater than or equal y (79ms)
    ✓ should fail if x is greater than or equal stakedFlexKeys' length  (78ms)
    ✓ should fail if y is greater than stakedFlexKeys' length  (83ms)
    valid case
      ✓ stakingFlex's amount info should be changed to the value added to the reward up to
the time the function is called
      ✓ stakingFlex's rate should be updated properly
      ✓ stakingFlex's timestamp should be updated properly (56ms)
  #increaseStakeFlex()
    ✓ should fail if msg.sender does not have a flexible staking
    ✓ should fail if contract's balance is less than msg.sender's total reward
    ✓ should fail if msg.sender's balance is less than amount (58ms)
    ✓ should fail if msg.sender's allowance for contract is less than amount (48ms)
    valid case
      ✓ contract's balance should increase
      ✓ msg.sender's balance should decrease
      ✓ balancesStaked should increase
      ✓ stakingFlex's amount should increase
      ✓ stakingFlex's timestamp should be updated (47ms)
      ✓ totalStaked should increase
      ✓ should emit Stake event
  #unstakeFlex()
    ✓ should fail if msg.sender does not have flexible staking
    ✓ should fail if amount exceeds staked balance
    ✓ should fail if amount is zero
    ✓ should fail if contract's balance is less than amount + reward (41ms)
    valid case
      ✓ stakingFlex's amount should decrease (51ms)
      ✓ stakingFlex's timestamp should be updated properly (51ms)
      ✓ balancesStaked should decrease (46ms)
      ✓ totalStake should decrease (47ms)
      ✓ contract's balance should decrease (46ms)
      ✓ msg.sender's balance should increase (48ms)
      ✓ should emit Unstake event (43ms)
    amount == stakedBalance case
      ✓ msg.sender's stakedFlexKey info should be deleted from stakedFlexKeys array (72ms)
      4) stakingFlex's ids should be updated properly
  #claimRewardFlex()
```

```
    ✓ should fail if msg.sender does not have flexible staking
    ✓ should fail contract's balance is less than reward (41ms)
    valid case
      ✓ stakingFlex's timestamp should be updated properly (50ms)
      ✓ contract's balance should decrease (42ms)
      ✓ msg.sender's balance should increase (43ms)
  #claimRewardLock()
    ✓ should fail if stake does not exist
    ✓ should fail if try to claim the reward before the end of the staking that is not claimable
(44ms)
    ✓ should fail if user has already claimed all the rewards (50ms)
    case1: the period of stakingPack has passed
      ✓ reward should be set properly with no penalty (63ms)
      ✓ should fail if contract's balance is less than reward (52ms)
      ✓ claimed amount should be equal to total reward (66ms)
      ✓ panaltyBalance should not be changed (66ms)
      ✓ contract's balance should decrease (46ms)
      ✓ msg.sender's balance should increase (47ms)
    case2: the period of stakingPack has not passed
      ✓ reward should be set properly with penalty (66ms)
      ✓ should fail if contract's balance is less than reward (44ms)
      ✓ claimed amount should increase (47ms)
      ✓ panaltyBalance should increase (51ms)
      ✓ contract's balance should decrease
      ✓ msg.sender's balance should increase
  #unstakeLock()
    ✓ should fail if stake does not exist
    ✓ should fail if contract's balance is less than amountUnstaked (50ms)
    ✓ balancesStaked should decrease (59ms)
    ✓ totalStake should decrease (63ms)
    ✓ currentUser should decrease (62ms)
    ✓ msg.sender's staked info should be deleted properly (61ms)
    ✓ contract's balance should decrease (58ms)
    ✓ msg.sender's balance should increase (59ms)
    ✓ should emit Unstake event (56ms)
    case1: the period of stakingPack has not passed
      ✓ should fail if stakingPack is not unlockable (45ms)
      ✓ amountUnstaked should be set properly with penalty (62ms)
      5) panaltyBalance should increase properly
    case2: the period of stakingPack has passed
      ✓ amountUnstaked should be set properly with reward (57ms)
      6) panaltyBalance should not change
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
MyTvFarming.sol	100	100	100	100	
MyTvGovernanceToken.sol	100	100	100	100	
MyTvLock.sol	100	100	100	100	
MyTvStaking.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document