Kommunitas

Smart Contract Security Audit

Prepared by ShellBoxes

July 20th, 2021 - July 27th, 2021

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Kommunitas
Target	Kommunitas Staking Smart Contract
Version	1.0
Classification	public

Scope

Repo	Commit Hash
https://github.com/Kommunitas-net/KommunitasToken	892ebb67592e195c90f5d45d88a6187de76539a6

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Kommunitas engaged ShellBoxes to conduct a security assessment on the Kommunitas beginning on July 20th, 2021 and ending July 27th, 2021. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Kommunitas

Kommunitas is a decentralized and tier-less Launchpad on Polygon. They are bridging the world to the biggest project in the most economical chain on cryptocurrency space. Kommunitas platform's goal is to allow project teams to focus on their project development and building their products, while the community handle the marketing, exposure and initial user base. They are looking for strong team with a unique and innovative vision in the cryptocurrency industry.

Issuer	Kommunitas
Website	`https://kommunitas.net`
Type	Polygon Smart Contract
Platform	Solidity
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's

scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Kommunitas implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 2 high-severity, 4 medium-severity, 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Burning Tokens Without Intention of User	HIGH	Fixed
Reentrancy Attack & Possibility of asynchronization in the communityStacked variable	HIGH	Fixed
Missing Address Validation	MEDIUM	Fixed
Owner can Renounce Ownership	MEDIUM	Acknowledged
For Loop Over Dynamic Array	MEDIUM	Acknowledged
Divide Before Multiply	MEDIUM	Fixed
Lack of verification in the constructor function	LOW	Fixed
Usage of Block.TimeStamp	LOW	Acknowledged
Floating Pragma	LOW	Fixed

3 Finding Details

A KommunitasStaking.sol

A.1 Burning Tokens Without Intention of User [HIGH]

Description:

The user can unlock the tokens that are staked if the maturity condition is verified, then the reward is automatically calculated and the komTokens are transferred to the address, if the staked tokens are less than the value 3000*1e8, a komvToken is automatically burned. The problem here is that any user can call this function and trigger this process, so inserting an address of a person who validates these conditions will cause his komvToken to be burned without having his permission.

Code:

Listing 1: KommunitasStaking.sol

```
121  if(getUserStakedTokens(_of) < 3000*1e8 && komvToken.balanceOf(_of) > 0)
       ↪ {
122    komvToken.burn(_of, 1);
123    }
```

Risk Level:

Likelihood – 5
Impact - 3

Recommendation:

Restrict the call of this function to the person who staked the tokens through a require and compare the msg.sender with the _of address, or modify the code so that the function uses the msg.sender variable directly.

Listing 2: KommunitasStaking.sol

```
105 function unlock(address _of) external returns (uint256) {
106     require(msg.sender == _of,"You can't call this function ! ");
107     uint256 unlockableTokens;
108     uint256 unlockablePrincipalStakedAmount;
```

Status - Fixed

KommunitasTeam has solved this issue by using the msg.sender as the address _of in commit 17ce810.

A.2 Reentrancy Attack & Possibility of asynchronization in the communityStacked variable [HIGH]

Description:

After verifying that all necessary conditions have been met, the contract sends the komTokens to the specified address, and this amount is then deducted from the variable communityStacked. The issue arises at the transfer function level, which does not verify if the transaction was properly completed, allowing a hacker to create a contract that forces the transaction to fail. However, the smart contract will receive the total amount of tokens but the instruction .sub will never be executed, and so the variable communityStaked will remain unchanged.

Code:

Listing 3: KommunitasStaking.sol

```
117 if (unlockableTokens > 0) {
118     komToken.transfer(_of, unlockableTokens);
119     communityStaked = communityStaked.sub(
        ↪ unlockablePrincipalStakedAmount);
```

Listing 4: KommunitasStaking.sol

```
156 komToken.transfer(msg.sender, withdrawableAmount);
157 communityStaked = communityStaked.sub(unlockableTokens);
158 if(getUserStakedTokens(msg.sender) < 3000*1e8 && komvToken.balanceOf(msg
       ↪ .sender) > 0){
159  komvToken.burn(msg.sender, 1);
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Always check if the transaction has not failed or any call of some external functions like transfer should be done last to avoid re-entrancy and synchronization problems.

Listing 5: KommunitasStaking.sol

```
113 if (unlockableTokens > 0) {
114  communityStaked = communityStaked.sub(unlockablePrincipalStakedAmount);
115  if(getUserStakedTokens(_of) < 3000*1e8 && komvToken.balanceOf(_of) > 0)
        ↪ {
116  komvToken.burn(_of, 1);
117  }
118  komToken.transfer(_of, unlockableTokens);
119  emit Unlocked(_of, unlockableTokens);
120  }
121  return unlockableTokens;
```

Listing 6: KommunitasStaking.sol

```
152 communityStaked = communityStaked.sub(unlockableTokens);
153 if(getUserStakedTokens(msg.sender) < 3000*1e8 && komvToken.balanceOf(msg
       ↪ .sender) > 0){
154  komvToken.burn(msg.sender, 1);
```

```
155 }
156 komToken.transfer(msg.sender, withdrawableAmount);
```

Status - Fixed

External calls have been moved to the end of the functions by the Kommunitas Team in commit 17ce810.

A.3 Missing Address Validation [MEDIUM]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible.

Code:

Listing 7: KommunitasStaking.sol

```
86 function _stake(address _user, uint256 _amount, uint256 _duration)
       ↪ internal {
87  require(_amount != 0, "Amount must not be zero.");
88  require(_duration <= maxDuration, "Lock exceeds maximum duration.");
89  require(_duration >= minDuration, "Locking period is too short.");
90  }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

It's recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Status - Fixed

The Kommunitas Team solved this issue by adding a verification in the _user address.

Listing 8: KommunitasStaking.sol

```
86 function _stake(address _user, uint256 _amount, uint256 _duration)
       ↪ internal {
87  require(_user != address(0), "Zero Address");
88  require(_amount != 0, "Amount must not be zero.");
89  require(_duration <= maxDuration, "Lock exceeds maximum duration.");
90  require(_duration >= minDuration, "Locking period is too short.");
91  }
```

A.4 Owner can Renounce Ownership [MEDIUM]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner.

Code:

Listing 9: KommunitasStaking.sol

```
10 contract KommunitasStaking is Ownable {
11  using SafeMath for uint256;
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Listing 10: KommunitasStaking.sol

```
function renounceOwnership() public override onlyOwner {
 revert("Impossible Action !");
}
```

Status - Acknowledged

Kommunitas team accepted this risk since to exploit this bug an attacker should control the Wallet of the Owner.

A.5 For Loop Over Dynamic Array [MEDIUM]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 11: KommunitasStaking.sol

```
110 for (uint256 i = 0; i < locksLength; i++) {
111     if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].
        ↪ claimed) {
```

Listing 12: KommunitasStaking.sol

```
135 for (uint256 i = 0; i < locksLength; i++) {
136  if (!locks[msg.sender][i].claimed) {
137  unlockableTokens = unlockableTokens.add(locks[msg.sender][i].amount);
```

Listing 13: KommunitasStaking.sol

```
173 for (uint256 i = 0; i < locksLength; i++) {
174  if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed
        ↪ ) {
175  withdrawableTokens = withdrawableTokens.add(locks[_of][i].amount).add(
        ↪ locks[_of][i]
```

Listing 14: KommunitasStaking.sol

```
204  for (uint256 i = 0; i < locksLength; i++) {
205  if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed
        ↪ ) {
```

Listing 15: KommunitasStaking.sol

```
188 for (uint256 i = 0; i < locksLength; i++) {
189  if (locks[_of][i].maturity > block.timestamp && !locks[_of][i].claimed)
        ↪  {
190  lockedTokens = lockedTokens.add(locks[_of][i].amount).add(locks[_of][i
        ↪ ].reward);
```

Listing 16: KommunitasStaking.sol

```
219 for (uint256 i = 0; i < locksLength; i++) {
220  if (!locks[_of][i].claimed) {
221  lockedTokens = lockedTokens.add(locks[_of][i].amount);
```

Risk Level:

Likelihood – 3

Impact - 2

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status - Acknowledged

The Kommunitas team accepted this risk.

A.6 Divide Before Multiply [MEDIUM]

Description:

Integer division in solidity may truncate. As a result, dividing before multiplying may result in a loss of precision. Due to precision's sensitivity, this may result in certain abnormalities in the contract's logic.

Code:

Listing 17: KommunitasStaking.sol

```
146 // User redeem penaltyFeesPercentage of there Unstake Amount
147 withdrawableAmount = (unlockableTokens.div(100)).mul(
        ↪ remainingFeePercentage);
148 // remaining is treated as penaltyAmount
149 uint256 penaltyAmount = (unlockableTokens.div(100)).mul(
        ↪ penaltyFeesPercentage);
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Do the multiplication operations before the division operations

Listing 18: KommunitasStaking.sol

```
146 // User redeem penaltyFeesPercentage of there Unstake Amount
147 withdrawableAmount = (unlockableTokens. mul(remainingFeePercentage).div
        ↪ (100));
148 // remaining is treated as penaltyAmount
149 uint256 penaltyAmount = (unlockableTokens.mul(penaltyFeesPercentage).div
        ↪ (100));
```

Status - Fixed

In the Staking Contract, the multiplication operation is performed before division in commit 17ce810.

A.7 Lack of verification in the constructor function [LOW]

Description:

In the constructor, the person who deployed the contract can add several parameters and among these parameters the variables minDuration and maxDuration. No verification is done for these variables and the creator of the contract can insert a value minDuration greater than maxDuration which will affect the logic of the contract.

Code:

Listing 19: KommunitasStaking.sol (Lines 48,49)

```
1 constructor(address _komToken, uint256 _apy, uint256 _minDuration,
      ↪ uint256 _maxDuration) {
2  komToken = ERC20Burnable(_komToken);
3  komvToken = new KommunitasVoting(); // KOM Governance Token Deployment
4  apy = _apy;
5  minDuration = _minDuration;
```

```
6   maxDuration = _maxDuration;
7   }
```

Risk Level:

Likelihood – 1
Impact - 4

Recommendation:

Add a condition to check that minDuration is smaller than maxDuration.

Listing 20: KommunitasStaking.sol

```
44  constructor(address _komToken, uint256 _apy, uint256 _minDuration,
        ↪ uint256 _maxDuration) {
45  require(_minDuration < _maxDuration,"MinDuration Should be Less than
        ↪ MaxDuration");
46  komToken = ERC20Burnable(_komToken);
47  komvToken = new KommunitasVoting(); // KOM Governance Token Deployment
```

Status - Fixed

Kommunitas Team has added the verification of the _minDuration and _maxDuration in commit 17ce810.

A.8 Usage of Block.TimeStamp [LOW]

Description:

block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Code:

Listing 21: KommunitasStaking.sol

```
91  uint256 matureUntil = block.timestamp.add(_duration);
92  uint256 lockReward = _calculateReward(_amount, _duration);
```

Listing 22: KommunitasStaking.sol

```
110 for (uint256 i = 0; i < locksLength; i++) {
111  if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed
        ↪ ) {
112  unlockableTokens = unlockableTokens.add(locks[_of][i].amount).add(locks
        ↪ [_of][i].reward ;
```

Listing 23: KommunitasStaking.sol

```
174 for (uint256 i = 0; i < locksLength; i++) {
175  if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed
        ↪ ) {
176  withdrawableTokens = withdrawableTokens.add(locks[_of][i].amount).add(
        ↪ locks[_of][i].rewar
```

Listing 24: KommunitasStaking.sol

```
189 for (uint256 i = 0; i < locksLength; i++) {
190  if (locks[_of][i].maturity > block.timestamp && !locks[_of][i].claimed)
        ↪  {
```

Listing 25: KommunitasStaking.sol

```
206 if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed)
       ↪  {
207  pendingRewards = pendingRewards.add(locks[_of][i].reward);
```

Risk Level:

Likelihood – 2

Impact - 2

Recommendation:

You can use an Oracle to get the exact time or verify if a delay of 900 seconds will not destroy the logic of the staking contract.

Status - Acknowledged

Kommunitas Team accepted this risk since 900 seconds will not affect the logic of the contract.

A.9 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.7.6. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps to ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 26: KommunitasStaking.sol

```
3 pragma solidity ^0.7.6;
4 import "@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol";
```

Risk Level:

Likelihood – 2
Impact - 1

Recommendation:

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

Kommunitas Team locked Pragma version to 0.7.6.

4 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
Compiled 13 Solidity files successfully

@openzeppelin/contracts/access/Ownable.sol:26:5: Warning: Visibility for
    ↪  constructor is ignored. If you want the contract to be non-
    ↪ deployable, making it "abstract" is sufficient.
    constructor () internal {
    ^ (Relevant source part starts here and spans across multiple lines)
        ↪ .

@openzeppelin/contracts/token/ERC20/ERC20.sol:55:5: Warning: Visibility
    ↪ for constructor is ignored. If you want the contract to be non-
    ↪ deployable, making it "abstract" is sufficient.
    constructor (string memory name_, string memory symbol_) public {
    ^ (Relevant source part starts here and spans across multiple lines)
        ↪ .

@openzeppelin/contracts/utils/ReentrancyGuard.sol:38:5: Warning:
    ↪ Visibility for constructor is ignored. If you want the contract
    ↪ to be non-deployable, making it "abstract" is sufficient.
    constructor () internal {
```

```
    ^ (Relevant source part starts here and spans across multiple lines)
      ↪ .



Reentrancy in OLDKommunitasStaking.exit() (contracts/
   ↪ OLDKommunitasStaking.sol#193-196):
 External calls:
 - unstake(getUserDetails[msg.sender].stakedAmount) (contracts/
    ↪ OLDKommunitasStaking.sol#194)
  - returndata = address(token).functionCall(data,SafeERC20: low-level
     ↪ call failed) (node_modules/@openzeppelin/contracts/token/ERC20/
     ↪ SafeERC20.sol#69)
  - (success,returndata) = target.call{value: value}(data) (node_modules
     ↪ /@openzeppelin/contracts/utils/Address.sol#119)
  - komToken.safeTransfer(userAccount,withdrawableAmount) (contracts/
     ↪ OLDKommunitasStaking.sol#163)
  - komToken.safeTransfer(owner(),rewardFees) (contracts/
     ↪ OLDKommunitasStaking.sol#164)
  - komToken.safeTransfer(userAccount,tokenAmount) (contracts/
     ↪ OLDKommunitasStaking.sol#168)
  - komvToken.burn(userAccount,1) (contracts/OLDKommunitasStaking.sol
     ↪ #172)
 - claimRewards() (contracts/OLDKommunitasStaking.sol#195)
  - returndata = address(token).functionCall(data,SafeERC20: low-level
     ↪ call failed) (node_modules/@openzeppelin/contracts/token/ERC20/
     ↪ SafeERC20.sol#69)
  - (success,returndata) = target.call{value: value}(data) (node_modules
     ↪ /@openzeppelin/contracts/utils/Address.sol#119)
  - komToken.safeTransfer(userAccount,reward) (contracts/
     ↪ OLDKommunitasStaking.sol#188)
 External calls sending eth:
 - unstake(getUserDetails[msg.sender].stakedAmount) (contracts/
    ↪ OLDKommunitasStaking.sol#194)
```

```
 - (success,returndata) = target.call{value: value}(data) (node_modules
    ↪ /@openzeppelin/contracts/utils/Address.sol#119)
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - (success,returndata) = target.call{value: value}(data) (node_modules
    ↪ /@openzeppelin/contracts/utils/Address.sol#119)
State variables written after the call(s):
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - _status = _ENTERED (node_modules/@openzeppelin/contracts/utils/
    ↪ ReentrancyGuard.sol#54)
 - _status = _NOT_ENTERED (node_modules/@openzeppelin/contracts/utils/
    ↪ ReentrancyGuard.sol#60)
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - getUserDetails[account].rewards = earned(account) (contracts/
    ↪ OLDKommunitasStaking.sol#52)
 - getUserDetails[account].userRewardPerTokenPaid =
    ↪ rewardPerTokenStored (contracts/OLDKommunitasStaking.sol#53)
 - getUserDetails[userAccount].rewards = 0 (contracts/
    ↪ OLDKommunitasStaking.sol#184)
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - lastUpdateTime = lastTimeRewardApplicable() (contracts/
    ↪ OLDKommunitasStaking.sol#50)
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - rewardPerTokenStored = rewardPerToken() (contracts/
    ↪ OLDKommunitasStaking.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities

KommunitasStaking._stake(address,uint256,uint256) (contracts/
   ↪ KommunitasStaking.sol#85-100) ignores return value by komToken.
   ↪ transferFrom(msg.sender,address(this),_amount) (contracts/
   ↪ KommunitasStaking.sol#93)
KommunitasStaking.unlock(address) (contracts/KommunitasStaking.sol
   ↪ #106-127) ignores return value by komToken.transfer(_of,
   ↪ unlockableTokens) (contracts/KommunitasStaking.sol#119)
```

```
KommunitasStaking.preMatureWithdraw() (contracts/KommunitasStaking.sol
    ↪ #132-164) ignores return value by komToken.transfer(msg.sender,
    ↪ withdrawableAmount) (contracts/KommunitasStaking.sol#156)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer

KommunitasStaking._calculateReward(uint256,uint256) (contracts/
    ↪ KommunitasStaking.sol#52-62) performs a multiplication on the
    ↪ result of a division:
 -effectiveAPY = multiplier.mul(apy).mul(durationSeconds).mul(1e10).div(
     ↪ yearDuration).div(10) (contracts/KommunitasStaking.sol#60)
 -_lockReward = effectiveAPY.mul(_amount).mul(durationSeconds).div(
     ↪ yearDuration).div(1e12) (contracts/KommunitasStaking.sol#61)
KommunitasStaking.preMatureWithdraw() (contracts/KommunitasStaking.sol
    ↪ #132-164) performs a multiplication on the result of a division:
 -withdrawableAmount = (unlockableTokens.div(100)).mul(
     ↪ remainingFeePercentage) (contracts/KommunitasStaking.sol#148)
KommunitasStaking.preMatureWithdraw() (contracts/KommunitasStaking.sol
    ↪ #132-164) performs a multiplication on the result of a division:
 -penaltyAmount = (unlockableTokens.div(100)).mul(penaltyFeesPercentage)
     ↪  (contracts/KommunitasStaking.sol#150)
OLDKommunitasStaking.unstake(uint256) (contracts/OLDKommunitasStaking.
    ↪ sol#145-176) performs a multiplication on the result of a
    ↪ division:
 -withdrawableAmount = (tokenAmount.div(100)).mul(remainingFeePercentage
     ↪ ) (contracts/OLDKommunitasStaking.sol#160)
OLDKommunitasStaking.unstake(uint256) (contracts/OLDKommunitasStaking.
    ↪ sol#145-176) performs a multiplication on the result of a
    ↪ division:
 -rewardFees = (tokenAmount.div(100)).mul(platformFeesPercentage) (
     ↪ contracts/OLDKommunitasStaking.sol#161)
OLDKommunitasStaking.notifyRewardAmount(uint256) (contracts/
    ↪ OLDKommunitasStaking.sol#200-218) performs a multiplication on
    ↪ the result of a division:
```

```
 -rewardRate = reward.div(rewardsDuration) (contracts/
    ↪ OLDKommunitasStaking.sol#206)
 -leftover = remaining.mul(rewardRate) (contracts/OLDKommunitasStaking.
    ↪ sol#209)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #divide-before-multiply

KommunitasStaking._stake(address,uint256,uint256) (contracts/
    ↪ KommunitasStaking.sol#85-100) uses a dangerous strict equality:
 - getUserStakedTokens(_user) > 3000 * 1e8 && komvToken.balanceOf(_user)
    ↪  == 0 (contracts/KommunitasStaking.sol#96)
KommunitasVoting._writeCheckpoint(address,uint32,uint256,uint256) (
    ↪ contracts/KommunitasVoting.sol#221-239) uses a dangerous strict
    ↪ equality:
 - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].
    ↪ fromBlock == blockNumber (contracts/KommunitasVoting.sol#231)
OLDKommunitasStaking.stake(uint256) (contracts/OLDKommunitasStaking.sol
    ↪ #119-143) uses a dangerous strict equality:
 - getUserDetails[userAccount].stakedAmount >= 3000 * 1e8 && komvToken.
    ↪ balanceOf(userAccount) == 0 (contracts/OLDKommunitasStaking.sol
    ↪ #138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dangerous-strict-equalities

Reentrancy in OLDKommunitasStaking.notifyRewardAmount(uint256) (
    ↪ contracts/OLDKommunitasStaking.sol#200-218):
 External calls:
 - komToken.safeTransferFrom(msg.sender,address(this),reward) (contracts
    ↪ /OLDKommunitasStaking.sol#212)
 State variables written after the call(s):
 - lastUpdateTime = block.timestamp (contracts/OLDKommunitasStaking.sol
    ↪ #215)
 - periodFinish = block.timestamp.add(rewardsDuration) (contracts/
    ↪ OLDKommunitasStaking.sol#216)
```

```
Reentrancy in OLDKommunitasStaking.stake(uint256) (contracts/
    ↪ OLDKommunitasStaking.sol#119-143):
 External calls:
 - komToken.safeTransferFrom(userAccount,address(this),tokenAmount) (
     ↪ contracts/OLDKommunitasStaking.sol#134)
 State variables written after the call(s):
 - getUserDetails[userAccount].exists = true (contracts/
     ↪ OLDKommunitasStaking.sol#136)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1

KommunitasStaking.getTotalWithdrawableTokens(address).withdrawableTokens
    ↪  (contracts/KommunitasStaking.sol#171) is a local variable never
    ↪ initialized
KommunitasStaking.getTotalLockedTokens(address).lockedTokens (contracts/
    ↪ KommunitasStaking.sol#186) is a local variable never initialized
KommunitasStaking.unlock(address).unlockableTokens (contracts/
    ↪ KommunitasStaking.sol#107) is a local variable never initialized
KommunitasStaking.getUserPendingRewards(address).pendingRewards (
    ↪ contracts/KommunitasStaking.sol#202) is a local variable never
    ↪ initialized
KommunitasStaking.preMatureWithdraw().unlockableTokens (contracts/
    ↪ KommunitasStaking.sol#133) is a local variable never initialized
KommunitasStaking.getUserStakedTokens(address).lockedTokens (contracts/
    ↪ KommunitasStaking.sol#217) is a local variable never initialized
KommunitasStaking.unlock(address).unlockablePrincipalStakedAmount (
    ↪ contracts/KommunitasStaking.sol#108) is a local variable never
    ↪ initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #uninitialized-local-variables

KommunitasStaking.updateLockDuration(uint256,uint256) (contracts/
    ↪ KommunitasStaking.sol#244-248) should emit an event for:
 - minDuration = _minDuration (contracts/KommunitasStaking.sol#246)
```

```
 - maxDuration = _maxDuration (contracts/KommunitasStaking.sol#247)
KommunitasStaking.updateAPY(uint256) (contracts/KommunitasStaking.sol
    ↪ #256-258) should emit an event for:
 - apy = _apy (contracts/KommunitasStaking.sol#257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic

Reentrancy in KommunitasStaking._stake(address,uint256,uint256) (
    ↪ contracts/KommunitasStaking.sol#85-100):
 External calls:
 - komToken.transferFrom(msg.sender,address(this),_amount) (contracts/
     ↪ KommunitasStaking.sol#93)
 State variables written after the call(s):
 - communityStaked = communityStaked.add(_amount) (contracts/
     ↪ KommunitasStaking.sol#95)
 - locks[_user].push(TokenLock(_amount,matureUntil,lockReward,false)) (
     ↪ contracts/KommunitasStaking.sol#94)
Reentrancy in OLDKommunitasStaking.notifyRewardAmount(uint256) (
    ↪ contracts/OLDKommunitasStaking.sol#200-218):
 External calls:
 - komToken.safeTransferFrom(msg.sender,address(this),reward) (contracts
     ↪ /OLDKommunitasStaking.sol#212)
 State variables written after the call(s):
 - totalRewardsReserve = totalRewardsReserve.add(reward) (contracts/
     ↪ OLDKommunitasStaking.sol#213)
Reentrancy in KommunitasStaking.preMatureWithdraw() (contracts/
    ↪ KommunitasStaking.sol#132-164):
 External calls:
 - komToken.burn(penaltyAmount) (contracts/KommunitasStaking.sol#153)
 - komToken.transfer(msg.sender,withdrawableAmount) (contracts/
     ↪ KommunitasStaking.sol#156)
 State variables written after the call(s):
 - communityStaked = communityStaked.sub(unlockableTokens) (contracts/
     ↪ KommunitasStaking.sol#157)
```

```
Reentrancy in KommunitasStaking.unlock(address) (contracts/
    ↪ KommunitasStaking.sol#106-127):
 External calls:
 - komToken.transfer(_of,unlockableTokens) (contracts/KommunitasStaking.
     ↪ sol#119)
 State variables written after the call(s):
 - communityStaked = communityStaked.sub(unlockablePrincipalStakedAmount
     ↪ ) (contracts/KommunitasStaking.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2

Reentrancy in KommunitasStaking._stake(address,uint256,uint256) (
    ↪ contracts/KommunitasStaking.sol#85-100):
 External calls:
 - komToken.transferFrom(msg.sender,address(this),_amount) (contracts/
     ↪ KommunitasStaking.sol#93)
 - komvToken.mint(_user,1) (contracts/KommunitasStaking.sol#97)
 Event emitted after the call(s):
 - Locked(_user,_amount,lockReward,matureUntil) (contracts/
     ↪ KommunitasStaking.sol#99)
Reentrancy in OLDKommunitasStaking.claimRewards() (contracts/
    ↪ OLDKommunitasStaking.sol#178-191):
 External calls:
 - komToken.safeTransfer(userAccount,reward) (contracts/
     ↪ OLDKommunitasStaking.sol#188)
 Event emitted after the call(s):
 - RewardsClaimed(userAccount,reward) (contracts/OLDKommunitasStaking.
     ↪ sol#190)
Reentrancy in OLDKommunitasStaking.exit() (contracts/
    ↪ OLDKommunitasStaking.sol#193-196):
 External calls:
 - unstake(getUserDetails[msg.sender].stakedAmount) (contracts/
     ↪ OLDKommunitasStaking.sol#194)
```

```
 - returndata = address(token).functionCall(data,SafeERC20: low-level
    ↪ call failed) (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ SafeERC20.sol#69)
 - (success,returndata) = target.call{value: value}(data) (node_modules
    ↪ /@openzeppelin/contracts/utils/Address.sol#119)
 - komToken.safeTransfer(userAccount,withdrawableAmount) (contracts/
    ↪ OLDKommunitasStaking.sol#163)
 - komToken.safeTransfer(owner(),rewardFees) (contracts/
    ↪ OLDKommunitasStaking.sol#164)
 - komToken.safeTransfer(userAccount,tokenAmount) (contracts/
    ↪ OLDKommunitasStaking.sol#168)
 - komvToken.burn(userAccount,1) (contracts/OLDKommunitasStaking.sol
    ↪ #172)
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - returndata = address(token).functionCall(data,SafeERC20: low-level
    ↪ call failed) (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ SafeERC20.sol#69)
 - (success,returndata) = target.call{value: value}(data) (node_modules
    ↪ /@openzeppelin/contracts/utils/Address.sol#119)
 - komToken.safeTransfer(userAccount,reward) (contracts/
    ↪ OLDKommunitasStaking.sol#188)
External calls sending eth:
- unstake(getUserDetails[msg.sender].stakedAmount) (contracts/
   ↪ OLDKommunitasStaking.sol#194)
 - (success,returndata) = target.call{value: value}(data) (node_modules
    ↪ /@openzeppelin/contracts/utils/Address.sol#119)
- claimRewards() (contracts/OLDKommunitasStaking.sol#195)
 - (success,returndata) = target.call{value: value}(data) (node_modules
    ↪ /@openzeppelin/contracts/utils/Address.sol#119)
Event emitted after the call(s):
- RewardsClaimed(userAccount,reward) (contracts/OLDKommunitasStaking.
   ↪ sol#190)
 - claimRewards() (contracts/OLDKommunitasStaking.sol#195)
```

```
Reentrancy in OLDKommunitasStaking.notifyRewardAmount(uint256) (
    ↪ contracts/OLDKommunitasStaking.sol#200-218):
 External calls:
 - komToken.safeTransferFrom(msg.sender,address(this),reward) (contracts
     ↪ /OLDKommunitasStaking.sol#212)
 Event emitted after the call(s):
 - RewarReservedAdded(reward) (contracts/OLDKommunitasStaking.sol#217)
Reentrancy in KommunitasStaking.preMatureWithdraw() (contracts/
    ↪ KommunitasStaking.sol#132-164):
 External calls:
 - komToken.burn(penaltyAmount) (contracts/KommunitasStaking.sol#153)
 - komToken.transfer(msg.sender,withdrawableAmount) (contracts/
     ↪ KommunitasStaking.sol#156)
 - komvToken.burn(msg.sender,1) (contracts/KommunitasStaking.sol#159)
 Event emitted after the call(s):
 - EmergencyUnlocked(msg.sender,unlockableTokens) (contracts/
     ↪ KommunitasStaking.sol#161)
Reentrancy in OLDKommunitasStaking.stake(uint256) (contracts/
    ↪ OLDKommunitasStaking.sol#119-143):
 External calls:
 - komToken.safeTransferFrom(userAccount,address(this),tokenAmount) (
     ↪ contracts/OLDKommunitasStaking.sol#134)
 - komvToken.mint(userAccount,1) (contracts/OLDKommunitasStaking.sol
     ↪ #139)
 Event emitted after the call(s):
 - Staked(userAccount,tokenAmount,block.timestamp,getUserDetails[
     ↪ userAccount].stakedAmount) (contracts/OLDKommunitasStaking.sol
     ↪ #142)
Reentrancy in KommunitasStaking.unlock(address) (contracts/
    ↪ KommunitasStaking.sol#106-127):
 External calls:
 - komToken.transfer(_of,unlockableTokens) (contracts/KommunitasStaking.
     ↪ sol#119)
 - komvToken.burn(_of,1) (contracts/KommunitasStaking.sol#122)
```

```
 Event emitted after the call(s):
 - Unlocked(_of,unlockableTokens) (contracts/KommunitasStaking.sol#124)
Reentrancy in OLDKommunitasStaking.unstake(uint256) (contracts/
    ↪ OLDKommunitasStaking.sol#145-176):
 External calls:
 - komToken.safeTransfer(userAccount,withdrawableAmount) (contracts/
     ↪ OLDKommunitasStaking.sol#163)
 - komToken.safeTransfer(owner(),rewardFees) (contracts/
     ↪ OLDKommunitasStaking.sol#164)
 - komToken.safeTransfer(userAccount,tokenAmount) (contracts/
     ↪ OLDKommunitasStaking.sol#168)
 - komvToken.burn(userAccount,1) (contracts/OLDKommunitasStaking.sol
     ↪ #172)
 Event emitted after the call(s):
 - Unstaked(userAccount,tokenAmount,block.timestamp,getUserDetails[
     ↪ userAccount].stakedAmount) (contracts/OLDKommunitasStaking.sol
     ↪ #175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3

KommunitasStaking.unlock(address) (contracts/KommunitasStaking.sol
    ↪ #106-127) uses timestamp for comparisons
 Dangerous comparisons:
 - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed
     ↪ (contracts/KommunitasStaking.sol#111)
KommunitasStaking.getTotalWithdrawableTokens(address) (contracts/
    ↪ KommunitasStaking.sol#170-179) uses timestamp for comparisons
 Dangerous comparisons:
 - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed
     ↪ (contracts/KommunitasStaking.sol#174)
KommunitasStaking.getTotalLockedTokens(address) (contracts/
    ↪ KommunitasStaking.sol#185-194) uses timestamp for comparisons
 Dangerous comparisons:
```

```
 - locks[_of][i].maturity > block.timestamp && ! locks[_of][i].claimed (
    ↪ contracts/KommunitasStaking.sol#189)
KommunitasStaking.getUserPendingRewards(address) (contracts/
   ↪ KommunitasStaking.sol#201-210) uses timestamp for comparisons
 Dangerous comparisons:
 - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed
    ↪ (contracts/KommunitasStaking.sol#205)
KommunitasVoting.delegateBySig(address,uint256,uint256,uint8,bytes32,
   ↪ bytes32) (contracts/KommunitasVoting.sol#87-128) uses timestamp
   ↪ for comparisons
 Dangerous comparisons:
 - require(bool,string)(block.timestamp <= expiry,KOMV::delegateBySig:
    ↪ signature expired) (contracts/KommunitasVoting.sol#126)
OLDKommunitasStaking.getCurrentRewardReserve() (contracts/
   ↪ OLDKommunitasStaking.sol#77-84) uses timestamp for comparisons
 Dangerous comparisons:
 - block.timestamp <= periodFinish (contracts/OLDKommunitasStaking.sol
    ↪ #78)
OLDKommunitasStaking.stake(uint256) (contracts/OLDKommunitasStaking.sol
   ↪ #119-143) uses timestamp for comparisons
 Dangerous comparisons:
 - require(bool,string)(block.timestamp <= periodFinish,Staking ended) (
    ↪ contracts/OLDKommunitasStaking.sol#124)
OLDKommunitasStaking.unstake(uint256) (contracts/OLDKommunitasStaking.
   ↪ sol#145-176) uses timestamp for comparisons
 Dangerous comparisons:
 - block.timestamp < periodFinish (contracts/OLDKommunitasStaking.sol
    ↪ #151)
OLDKommunitasStaking.claimRewards() (contracts/OLDKommunitasStaking.sol
   ↪ #178-191) uses timestamp for comparisons
 Dangerous comparisons:
 - require(bool,string)(block.timestamp > periodFinish && !
    ↪ getUserDetails[userAccount].preWithdrawal,Rewards can't be
    ↪ claimed due to pre-withdrawal) (contracts/OLDKommunitasStaking.
```

```
    ↪ sol#180)
 - require(bool,string)(reward > 0,No Claimable rewards pending) (
    ↪ contracts/OLDKommunitasStaking.sol#182)
OLDKommunitasStaking.notifyRewardAmount(uint256) (contracts/
   ↪ OLDKommunitasStaking.sol#200-218) uses timestamp for comparisons
 Dangerous comparisons:
 - block.timestamp >= periodFinish (contracts/OLDKommunitasStaking.sol
    ↪ #205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #block-timestamp

Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/
   ↪ Address.sol#26-35) uses assembly
 - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol
    ↪ #33)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin
   ↪ /contracts/utils/Address.sol#171-188) uses assembly
 - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol
    ↪ #180-183)
KommunitasVoting.getChainId() (contracts/KommunitasVoting.sol#246-250)
   ↪ uses assembly
 - INLINE ASM (contracts/KommunitasVoting.sol#248)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #assembly-usage

Different versions of Solidity is used:
 - Version used: ['>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.7.0', '^0.7.6']
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.
    ↪ sol#3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/Math.sol#3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol
    ↪ #3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20
    ↪ .sol#3)
```

```
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ ERC20Burnable.sol#3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ IERC20.sol#3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ SafeERC20.sol#3)
 - >=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol
    ↪ #3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
    ↪ #3)
 - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/
    ↪ ReentrancyGuard.sol#3)
 - ^0.7.6 (contracts/KommunitasStaking.sol#3)
 - ^0.7.6 (contracts/KommunitasVoting.sol#2)
 - ^0.7.0 (contracts/OLDKommunitasStaking.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access
    ↪ /Ownable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/
    ↪ Math.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/
    ↪ SafeMath.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/
    ↪ ERC20/ERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/
    ↪ ERC20/ERC20Burnable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/
    ↪ ERC20/IERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/token/
    ↪ ERC20/SafeERC20.sol#3) is too complex
Pragma version>=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts/utils/
    ↪ Address.sol#3) is too complex
```

```
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/
    ↪ Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/
    ↪ ReentrancyGuard.sol#3) is too complex
Pragma version^0.7.0 (contracts/OLDKommunitasStaking.sol#2) allows old
    ↪ versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/
    ↪ @openzeppelin/contracts/utils/Address.sol#53-59):
 - (success) = recipient.call{value: amount}() (node_modules/
     ↪ @openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,
    ↪ string) (node_modules/@openzeppelin/contracts/utils/Address.sol
    ↪ #114-121):
 - (success,returndata) = target.call{value: value}(data) (node_modules/
     ↪ @openzeppelin/contracts/utils/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (
    ↪ node_modules/@openzeppelin/contracts/utils/Address.sol#139-145):
 - (success,returndata) = target.staticcall(data) (node_modules/
     ↪ @openzeppelin/contracts/utils/Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (
    ↪ node_modules/@openzeppelin/contracts/utils/Address.sol#163-169):
 - (success,returndata) = target.delegatecall(data) (node_modules/
     ↪ @openzeppelin/contracts/utils/Address.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #low-level-calls

Parameter KommunitasStaking.lockedStake(uint256,uint256)._amount (
    ↪ contracts/KommunitasStaking.sol#70) is not in mixedCase
Parameter KommunitasStaking.lockedStake(uint256,uint256)._duration (
    ↪ contracts/KommunitasStaking.sol#70) is not in mixedCase
```

```
Parameter KommunitasStaking.delegateLockedStaking(address,uint256,
    ↪ uint256)._user (contracts/KommunitasStaking.sol#81) is not in
    ↪ mixedCase
Parameter KommunitasStaking.delegateLockedStaking(address,uint256,
    ↪ uint256)._amount (contracts/KommunitasStaking.sol#81) is not in
    ↪ mixedCase
Parameter KommunitasStaking.delegateLockedStaking(address,uint256,
    ↪ uint256)._duration (contracts/KommunitasStaking.sol#81) is not in
    ↪  mixedCase
Parameter KommunitasStaking.unlock(address)._of (contracts/
    ↪ KommunitasStaking.sol#106) is not in mixedCase
Parameter KommunitasStaking.getTotalWithdrawableTokens(address)._of (
    ↪ contracts/KommunitasStaking.sol#170) is not in mixedCase
Parameter KommunitasStaking.getTotalLockedTokens(address)._of (contracts
    ↪ /KommunitasStaking.sol#185) is not in mixedCase
Parameter KommunitasStaking.getUserPendingRewards(address)._of (
    ↪ contracts/KommunitasStaking.sol#201) is not in mixedCase
Parameter KommunitasStaking.getUserStakedTokens(address)._of (contracts/
    ↪ KommunitasStaking.sol#216) is not in mixedCase
Parameter KommunitasStaking.setDurationMultiplier(uint256,uint256).
    ↪ _multiplier (contracts/KommunitasStaking.sol#234) is not in
    ↪ mixedCase
Parameter KommunitasStaking.setDurationMultiplier(uint256,uint256).
    ↪ _duration (contracts/KommunitasStaking.sol#234) is not in
    ↪ mixedCase
Parameter KommunitasStaking.updateLockDuration(uint256,uint256).
    ↪ _minDuration (contracts/KommunitasStaking.sol#244) is not in
    ↪ mixedCase
Parameter KommunitasStaking.updateLockDuration(uint256,uint256).
    ↪ _maxDuration (contracts/KommunitasStaking.sol#244) is not in
    ↪ mixedCase
Parameter KommunitasStaking.updatePenaltyFees(uint256)._feesPercentage (
    ↪ contracts/KommunitasStaking.sol#250) is not in mixedCase
```

```
Parameter KommunitasStaking.updateAPY(uint256)._apy (contracts/
    ↪ KommunitasStaking.sol#256) is not in mixedCase
Constant KommunitasStaking.yearDuration (contracts/KommunitasStaking.sol
    ↪ #23) is not in UPPER_CASE_WITH_UNDERSCORES
Parameter KommunitasVoting.mint(address,uint256)._to (contracts/
    ↪ KommunitasVoting.sol#18) is not in mixedCase
Parameter KommunitasVoting.mint(address,uint256)._amount (contracts/
    ↪ KommunitasVoting.sol#18) is not in mixedCase
Parameter KommunitasVoting.burn(address,uint256)._to (contracts/
    ↪ KommunitasVoting.sol#23) is not in mixedCase
Parameter KommunitasVoting.burn(address,uint256)._amount (contracts/
    ↪ KommunitasVoting.sol#23) is not in mixedCase
Variable KommunitasVoting._delegates (contracts/KommunitasVoting.sol#29)
    ↪  is not in mixedCase
Parameter OLDKommunitasStaking.changeWithdrawalFees(uint256).
    ↪ _feesPercentage (contracts/OLDKommunitasStaking.sol#220) is not
    ↪ in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions

Redundant expression "this (node_modules/@openzeppelin/contracts/utils/
    ↪ Context.sol#21)" inContext (node_modules/@openzeppelin/contracts/
    ↪ utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #redundant-statements

renounceOwnership() should be declared external:
 - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/
    ↪ access/Ownable.sol#54-57)
transferOwnership(address) should be declared external:
 - Ownable.transferOwnership(address) (node_modules/@openzeppelin/
    ↪ contracts/access/Ownable.sol#63-67)
symbol() should be declared external:
```

```
 - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ ERC20.sol#72-74)
decimals() should be declared external:
 - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ ERC20.sol#89-91)
totalSupply() should be declared external:
 - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20
    ↪ /ERC20.sol#96-98)
transfer(address,uint256) should be declared external:
 - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts
    ↪ /token/ERC20/ERC20.sol#115-118)
approve(address,uint256) should be declared external:
 - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/
    ↪ token/ERC20/ERC20.sol#134-137)
transferFrom(address,address,uint256) should be declared external:
 - ERC20.transferFrom(address,address,uint256) (node_modules/
    ↪ @openzeppelin/contracts/token/ERC20/ERC20.sol#152-156)
increaseAllowance(address,uint256) should be declared external:
 - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts/token/ERC20/ERC20.sol#170-173)
decreaseAllowance(address,uint256) should be declared external:
 - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts/token/ERC20/ERC20.sol#189-192)
burn(uint256) should be declared external:
 - ERC20Burnable.burn(uint256) (node_modules/@openzeppelin/contracts/
    ↪ token/ERC20/ERC20Burnable.sol#21-23)
burnFrom(address,uint256) should be declared external:
 - ERC20Burnable.burnFrom(address,uint256) (node_modules/@openzeppelin/
    ↪ contracts/token/ERC20/ERC20Burnable.sol#36-41)
getTotalWithdrawableTokens(address) should be declared external:
 - KommunitasStaking.getTotalWithdrawableTokens(address) (contracts/
    ↪ KommunitasStaking.sol#170-179)
getTotalLockedTokens(address) should be declared external:
```

```
 - KommunitasStaking.getTotalLockedTokens(address) (contracts/
    ↪ KommunitasStaking.sol#185-194)
getUserPendingRewards(address) should be declared external:
 - KommunitasStaking.getUserPendingRewards(address) (contracts/
    ↪ KommunitasStaking.sol#201-210)
mint(address,uint256) should be declared external:
 - KommunitasVoting.mint(address,uint256) (contracts/KommunitasVoting.
    ↪ sol#18-21)
burn(address,uint256) should be declared external:
 - KommunitasVoting.burn(address,uint256) (contracts/KommunitasVoting.
    ↪ sol#23-26)
getCurrentRewardReserve() should be declared external:
 - OLDKommunitasStaking.getCurrentRewardReserve() (contracts/
    ↪ OLDKommunitasStaking.sol#77-84)
getCommunity() should be declared external:
 - OLDKommunitasStaking.getCommunity() (contracts/OLDKommunitasStaking.
    ↪ sol#110-112)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #public-function-that-could-be-declared-external
. analyzed (13 contracts with 77 detectors), 108 result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

In this audit, we examined the design and implementation of Kommunitas contract and discovered several issues of varying severity. Kommunitas team addressed 6 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Kommunitas Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

For a Contract Audit, contact us at contact@shellboxes.com