
# Kommunitas

## Smart Contract Security Audit

Prepared by: ShellBoxes

July 20, 2021 – July 27, 2021

Shellboxes.com

contact@shellboxes.com

# Document Properties

| | |
|---|---|
| Client | Kommunitas |
| Target | Kommunitas Staking Smart Contract |
| Version | 1.0 |
| Auditors | Farouk El Alem, Inas Hasnaoui |
| Reviewed By | Inas Hasnaoui |
| Approved By | Inas Hasnaoui |
| Classification | Public |

# Document History

| VERSION | MODIFICATION | DATE | AUTHOR |
|---|---|---|---|
| 0.1 | Initial Draft | July 23, 2021 | Farouk El Alem |
| 0.2 | Additional Findings | July 27, 2021 | Inas Hasnaoui |
| 0.3 | Final Version | August 01, 2021 | Inas Hasnaoui |
| 1.0 | Remediation Plan | August 02, 2021 | Farouk El Alem |

# Contacts

| VERSION | COMPANY | EMAIL |
|---|---|---|
| Inas Hasnaoui | ShellBoxes | i.hasnaoui@shellboxes.com |
| Farouk El Alem | ShellBoxes | f.elalem@shellboxes.com |

# Contents

# 1. Introduction

Kommunitas engaged ShellBoxes to conduct a security assessment on the Staking Smart Contract beginning on July 20th, 2021 and ending July 27th, 2021. We detail our methodical methodology in this report to evaluate potential security issues in the smart contract implementation, exposing possible semantic discrepancies between the smart contract code and design document, and making additional ideas or recommendations for improvement. Our findings indicate that the current version of smart contracts can 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.

## 1.2. Approach & Methodology

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

ShellBoxes used a combination of manual and automated security testing to strike a balance between efficiency, timeliness, practicability, and correctness in relation to 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 items that violate 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 | | |

## 1.3. Scope

The Staking Contract in the Kommunitas Repository

Commit ID: 892ebb67592e195c90f5d45d88a6187de76539a6

# 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 critical-severity vulnerability, 4 medium-severity vulnerability, 3 low-severity vulnerabilities.

| Vulnerabilities | Severity | Status |
|---|---|---|
| Burning Tokens Without Intention of User | High | Fixed |
| Re-Entrancy Attack | 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 |
| Floating Pragma | Low | Fixed |
| Usage of Block.TimeStamp | Low | Acknowledged |

# 3. Findings Details

## 3.1. 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 (Lines 121,122)

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

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.

Fix 1 : KommunitasStaking (Lines 106)

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

Solved: Kommunitas Team has solved this issue by using the msg.sender as the address _of in commit 17ce810 .

## 3.2. 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 2 : KommunitasStaking (Lines 118)

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

Listing 3 : KommunitasStaking (Lines 156)

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

Fix 2 : KommunitasStaking (Lines 118)

```
        if (unlockableTokens > 0) {
             communityStaked = communityStaked.sub(unlockablePrincipalStakedAmount);
             if(getUserStakedTokens(_of) < 3000*1e8 && komvToken.balanceOf(_of) > 0){
                 komvToken.burn(_of, 1);
             }
             komToken.transfer(_of, unlockableTokens);
             emit Unlocked(_of, unlockableTokens);
         }
         return unlockableTokens;
```

Fix 3 : KommunitasStaking (Lines 156)

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

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

## 3.3. 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 or tokens may be burned in perpetuity.

Code

Listing 4 : KommunitasStaking (Lines 86, 87)

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

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.

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

Fixed 1 : KommunitasStaking (Lines 86, 87)

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

## 3.4. 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, which is risky.

Listing 5 : KommunitasStaking (Lines 10, 11)

```
contract KommunitasStaking is Ownable {
    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.

Fix 4 : KommunitasStaking (Lines -)

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

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

## 3.5. 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 attack.

Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 6 : KommunitasStaking (Lines 110,111)

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

Listing 7 : KommunitasStaking (Lines 135,136)

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

Listing 8 : KommunitasStaking (Lines 173,174)

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

Listing 9 : KommunitasStaking (Lines 204,205)

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

Listing 10 : KommunitasStaking (Lines 188,189)

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

Listing 11 : KommunitasStaking (Lines 219,220,221)

```
        for (uint256 i = 0; i < locksLength; i++) {
            if (!locks[_of][i].claimed) {
                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.

Risk Accepted: Kommunitas team accepted this risk.

## 3.6. 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 12 : KommunitasStaking (Lines 147)

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

Risk Level:

Likelihood – 1

Impact - 2

Recommendation:

Do the multiplication operations before the division operations

Fix 5 : KommunitasStaking (Lines 147)

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

Solved: In the Staking Contract, the Multiplication operation is performed before division in commit 17ce810.

## 3.7. 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 13: KommunitasStaking.sol (Lines 48,49)

```
constructor(address _komToken, uint256 _apy, uint256 _minDuration, uint256 _maxDuration) {
    komToken = ERC20Burnable(_komToken);
    komvToken = new KommunitasVoting(); // KOM Governance Token Deployment
    apy = _apy;
    minDuration = _minDuration;
    maxDuration = _maxDuration;
}
```

Risk Level:

Likelihood – 1

Impact – 4

Recommendation:

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

Fix 6: KommunitasStaking.sol (Lines 48,49)

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

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

## 3.8. 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 14 : KommunitasStaking (Lines 91, 92)

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

Listing 15 : KommunitasStaking (Lines 111, 113)

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

Listing 16 : KommunitasStaking (Lines 174, 176)

```
for (uint256 i = 0; i < locksLength; i++) {

    if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed) {

        withdrawableTokens = withdrawableTokens.add(locks[_of][i].amount).add(locks[_of][i].rewar
```

Listing 17 : KommunitasStaking (Lines 189, 190)

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

Listing 18 : KommunitasStaking (Lines 206, 207)

```
            if (locks[_of][i].maturity <= block.timestamp && !locks[_of][i].claimed) {
                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 won't destroy the logic of the staking contract.

Risk Accepted: Kommunitas Team accepted this risk since 900 seconds won't affect the logic of the contract.

## 3.9. 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 ensuring that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Code:

Listing 19 : KommunitasStaking (Lines 3, 4)

```
pragma solidity ^0.7.6;
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.

Solved: Kommunitas Team locked Pragma version to 0.7.6

## 3.10. Static Analysis

### Description:

ShellBoxes augmented coverage of the specific contract areas through the use of 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:

```
KommunitasStaking._stake(address,uint256,uint256) (KommunitasStaking.sol#85-100) ignores return
value by komToken.transferFrom(msg.sender,address(this),_amount) (KommunitasStaking.sol#93)
KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127) ignores return value by komTok
en.transfer(_of,unlockableTokens) (KommunitasStaking.sol#119)
KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164) ignores return value by kc
mToken.transfer(msg.sender,withdrawableAmount) (KommunitasStaking.sol#156)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
KommunitasStaking._calculateReward(uint256,uint256) (KommunitasStaking.sol#52-62) performs a mul
tiplication on the result of a division:
        -effectiveAPY = multiplier.mul(apy).mul(durationSeconds).mul(1e10).div(yearDuration).div
(10) (KommunitasStaking.sol#60)
        -_lockReward = effectiveAPY.mul(_amount).mul(durationSeconds).div(yearDuration).div(1e12
) (KommunitasStaking.sol#61)
KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164) performs a multiplication
on the result of a division:
        -withdrawableAmount = (unlockableTokens.div(100)).mul(remainingFeePercentage) (Kommunita
sStaking.sol#148)
KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164) performs a multiplication
on the result of a division:
        -penaltyAmount = (unlockableTokens.div(100)).mul(penaltyFeesPercentage) (KommunitasStaki
ng.sol#150)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
KommunitasStaking._stake(address,uint256,uint256) (KommunitasStaking.sol#85-100) uses a dangerou
s strict equality:
        - getUserStakedTokens(_user) > 3000 * 1e8 && komvToken.balanceOf(_user) == 0 (Kommunitas
Staking.sol#96)
KommunitasVoting._writeCheckpoint(address,uint32,uint256,uint256) (KommunitasVoting.sol#219-237)
 uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber
(KommunitasVoting.sol#229)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equali
ties
INFO:Detectors:
KommunitasStaking.preMatureWithdraw().unlockableTokens (KommunitasStaking.sol#133) is a local va
riable never initialized
KommunitasStaking.getTotalWithdrawableTokens(address).withdrawableTokens (KommunitasStaking.sol#
171) is a local variable never initialized
KommunitasStaking.unlock(address).unlockableTokens (KommunitasStaking.sol#107) is a local variab
le never initialized
```

```
KommunitasStaking.unlock(address).unlockablePrincipalStakedAmount (KommunitasStaking.sol#108) is
 a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-var
iables
INFO:Detectors:
KommunitasStaking.updateLockDuration(uint256,uint256) (KommunitasStaking.sol#244-248) should emi
t an event for:
        - minDuration = _minDuration (KommunitasStaking.sol#246)
        - maxDuration = _maxDuration (KommunitasStaking.sol#247)
KommunitasStaking.updateAPY(uint256) (KommunitasStaking.sol#256-258) should emit an event for:
        - apy = _apy (KommunitasStaking.sol#257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmet
ic
INFO:Detectors:
Reentrancy in KommunitasStaking._stake(address,uint256,uint256) (KommunitasStaking.sol#85-100):
        External calls:
        - komToken.transferFrom(msg.sender,address(this),_amount) (KommunitasStaking.sol#93)
        State variables written after the call(s):
        - communityStaked = communityStaked.add(_amount) (KommunitasStaking.sol#95)
        - locks[_user].push(TokenLock(_amount,matureUntil,lockReward,false)) (KommunitasStaking.
sol#94)
Reentrancy in KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164):
        External calls:
        - komToken.burn(penaltyAmount) (KommunitasStaking.sol#153)
        - komToken.transfer(msg.sender,withdrawableAmount) (KommunitasStaking.sol#156)
        State variables written after the call(s):
        - communityStaked = communityStaked.sub(unlockableTokens) (KommunitasStaking.sol#157)
Reentrancy in KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127):
        External calls:
        - komToken.transfer(_of,unlockableTokens) (KommunitasStaking.sol#119)
        State variables written after the call(s):
        - communityStaked = communityStaked.sub(unlockablePrincipalStakedAmount) (KommunitasStak
ing.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilit
ies-2
INFO:Detectors:
Reentrancy in KommunitasStaking._stake(address,uint256,uint256) (KommunitasStaking.sol#85-100):
        External calls:
        - komToken.transferFrom(msg.sender,address(this),_amount) (KommunitasStaking.sol#93)
        - komvToken.mint(_user,1) (KommunitasStaking.sol#97)
        Event emitted after the call(s):
        - Locked(_user,_amount,lockReward,matureUntil) (KommunitasStaking.sol#99)
Reentrancy in KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164):
```

```
        External calls:
        - komToken.burn(penaltyAmount) (KommunitasStaking.sol#153)
        - komToken.transfer(msg.sender,withdrawableAmount) (KommunitasStaking.sol#156)
        State variables written after the call(s):
        - communityStaked = communityStaked.sub(unlockableTokens) (KommunitasStaking.sol#157)
Reentrancy in KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127):
        External calls:
        - komToken.transfer(_of,unlockableTokens) (KommunitasStaking.sol#119)
        State variables written after the call(s):
        - communityStaked = communityStaked.sub(unlockablePrincipalStakedAmount) (KommunitasStak
ing.sol#120)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilit
ies-2
INFO:Detectors:
Reentrancy in KommunitasStaking._stake(address,uint256,uint256) (KommunitasStaking.sol#85-100):
        External calls:
        - komToken.transferFrom(msg.sender,address(this),_amount) (KommunitasStaking.sol#93)
        - komvToken.mint(_user,1) (KommunitasStaking.sol#97)
        Event emitted after the call(s):
        - Locked(_user,_amount,lockReward,matureUntil) (KommunitasStaking.sol#99)
Reentrancy in KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164):
        External calls:
        - komToken.burn(penaltyAmount) (KommunitasStaking.sol#153)
        - komToken.transfer(msg.sender,withdrawableAmount) (KommunitasStaking.sol#156)
        - komvToken.burn(msg.sender,1) (KommunitasStaking.sol#159)
        Event emitted after the call(s):
        - EmergencyUnlocked(msg.sender,unlockableTokens) (KommunitasStaking.sol#161)
Reentrancy in KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127):
        External calls:
        - komToken.transfer(_of,unlockableTokens) (KommunitasStaking.sol#119)
        - komvToken.burn(_of,1) (KommunitasStaking.sol#122)
        Event emitted after the call(s):
        - Unlocked(_of,unlockableTokens) (KommunitasStaking.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilit
ies-3
INFO:Detectors:
KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127) uses timestamp for comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed (KommunitasStakin
g.sol#111)
KommunitasStaking.getTotalWithdrawableTokens(address) (KommunitasStaking.sol#170-179) uses times
tamp for comparisons
```

```
        - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed (KommunitasStakin
g.sol#174)
KommunitasStaking.getTotalLockedTokens(address) (KommunitasStaking.sol#185-194) uses timestamp f
or comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity > block.timestamp && ! locks[_of][i].claimed (KommunitasStaking
.sol#189)
KommunitasStaking.getUserPendingRewards(address) (KommunitasStaking.sol#201-210) uses timestamp
for comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed (KommunitasStakin
g.sol#205)
KommunitasVoting.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (KommunitasVoting.
sol#85-126) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,KOMV::delegateBySig: signature expired)
 (KommunitasVoting.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (../OpenZeppelin/utils/Address.sol#26-36) uses assembly
        - INLINE ASM (../OpenZeppelin/utils/Address.sol#32-34)
Address.verifyCallResult(bool,bytes,string) (../OpenZeppelin/utils/Address.sol#195-215) uses ass
embly
        - INLINE ASM (../OpenZeppelin/utils/Address.sol#207-210)
KommunitasVoting.getChainId() (KommunitasVoting.sol#244-248) uses assembly
        - INLINE ASM (KommunitasVoting.sol#246)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (../OpenZeppelin/utils/Address.sol#79-81) is never used and
should be removed
Address.functionCall(address,bytes,string) (../OpenZeppelin/utils/Address.sol#89-95) is never us
ed and should be removed
Address.functionCallWithValue(address,bytes,uint256) (../OpenZeppelin/utils/Address.sol#108-114)
 is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (../OpenZeppelin/utils/Address.sol#1
22-133) is never used and should be removed
Address.functionDelegateCall(address,bytes) (../OpenZeppelin/utils/Address.sol#168-170) is never
 used and should be removed
Address.functionDelegateCall(address,bytes,string) (../OpenZeppelin/utils/Address.sol#178-187) i
s never used and should be removed
Address.functionStaticCall(address,bytes) (../OpenZeppelin/utils/Address.sol#141-143) is never u
sed and should be removed
```

```
Address.functionDelegateCall(address,bytes,string) (../OpenZeppelin/utils/Address.sol#178-187) i
s never used and should be removed
Address.functionStaticCall(address,bytes) (../OpenZeppelin/utils/Address.sol#141-143) is never u
sed and should be removed
Address.functionStaticCall(address,bytes,string) (../OpenZeppelin/utils/Address.sol#151-160) is
never used and should be removed
Address.isContract(address) (../OpenZeppelin/utils/Address.sol#26-36) is never used and should b
e removed
Address.sendValue(address,uint256) (../OpenZeppelin/utils/Address.sol#54-59) is never used and s
hould be removed
Address.verifyCallResult(bool,bytes,string) (../OpenZeppelin/utils/Address.sol#195-215) is never
 used and should be removed
Context._msgData() (../OpenZeppelin/utils/Context.sol#20-22) is never used and should be removed
Math.average(uint256,uint256) (../OpenZeppelin/utils/math/Math.sol#27-30) is never used and shou
ld be removed
Math.ceilDiv(uint256,uint256) (../OpenZeppelin/utils/math/Math.sol#38-41) is never used and shou
ld be removed
Math.max(uint256,uint256) (../OpenZeppelin/utils/math/Math.sol#12-14) is never used and should b
e removed
Math.min(uint256,uint256) (../OpenZeppelin/utils/math/Math.sol#19-21) is never used and should b
e removed
SafeMath.div(uint256,uint256,string) (../OpenZeppelin/utils/math/SafeMath.sol#190-199) is never
used and should be removed
SafeMath.mod(uint256,uint256) (../OpenZeppelin/utils/math/SafeMath.sol#150-152) is never used an
d should be removed
SafeMath.mod(uint256,uint256,string) (../OpenZeppelin/utils/math/SafeMath.sol#216-225) is never
used and should be removed
SafeMath.sub(uint256,uint256,string) (../OpenZeppelin/utils/math/SafeMath.sol#167-176) is never
used and should be removed
SafeMath.tryAdd(uint256,uint256) (../OpenZeppelin/utils/math/SafeMath.sol#21-27) is never used a
nd should be removed
SafeMath.tryDiv(uint256,uint256) (../OpenZeppelin/utils/math/SafeMath.sol#63-68) is never used a
nd should be removed
SafeMath.tryMod(uint256,uint256) (../OpenZeppelin/utils/math/SafeMath.sol#75-80) is never used a
nd should be removed
SafeMath.tryMul(uint256,uint256) (../OpenZeppelin/utils/math/SafeMath.sol#46-56) is never used a
nd should be removed
SafeMath.trySub(uint256,uint256) (../OpenZeppelin/utils/math/SafeMath.sol#34-39) is never used a
nd should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (../OpenZeppelin/access/Ownable.sol#3) necessitates a version too recent to
 be trusted. Consider deploying with 0.6.12/0.7.6
```

```
Pragma version^0.8.0 (../OpenZeppelin/token/ERC20/ERC20.sol#3) necessitates a version too recent
 to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/token/ERC20/IERC20.sol#3) necessitates a version too recen
t to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/token/ERC20/extensions/ERC20Burnable.sol#3) necessitates a
 version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/token/ERC20/extensions/IERC20Metadata.sol#3) necessitates
a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/utils/Address.sol#3) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/utils/Context.sol#3) necessitates a version too recent to
be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/utils/math/Math.sol#3) necessitates a version too recent t
o be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../OpenZeppelin/utils/math/SafeMath.sol#3) necessitates a version too rece
nt to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (KommunitasStaking.sol#3) necessitates a version too recent to be trusted.
Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (KommunitasVoting.sol#2) necessitates a version too recent to be trusted. C
onsider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-s
olidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (../OpenZeppelin/utils/Address.sol#54-59):
        - (success) = recipient.call{value: amount}() (../OpenZeppelin/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (../OpenZeppelin/u
tils/Address.sol#122-133):
        - (success,returndata) = target.call{value: value}(data) (../OpenZeppelin/utils/Address.
sol#131)
Low level call in Address.functionStaticCall(address,bytes,string) (../OpenZeppelin/utils/Addres
s.sol#151-160):
        - (success,returndata) = target.staticcall(data) (../OpenZeppelin/utils/Address.sol#158)
Low level call in Address.functionDelegateCall(address,bytes,string) (../OpenZeppelin/utils/Addr
ess.sol#178-187):
        - (success,returndata) = target.delegatecall(data) (../OpenZeppelin/utils/Address.sol#18
5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter KommunitasStaking.lockedStake(uint256,uint256)._amount (KommunitasStaking.sol#70) is n
ot in mixedCase
Parameter KommunitasStaking.lockedStake(uint256,uint256)._duration (KommunitasStaking.sol#70) is
 not in mixedCase
```

```
Parameter KommunitasStaking.delegateLockedStaking(address,uint256,uint256)._user (KommunitasStak
ing.sol#81) is not in mixedCase
Parameter KommunitasStaking.delegateLockedStaking(address,uint256,uint256)._amount (KommunitasSt
aking.sol#81) is not in mixedCase
Parameter KommunitasStaking.delegateLockedStaking(address,uint256,uint256)._duration (Kommunitas
Staking.sol#81) is not in mixedCase
Parameter KommunitasStaking.unlock(address)._of (KommunitasStaking.sol#106) is not in mixedCase
Parameter KommunitasStaking.getTotalWithdrawableTokens(address)._of (KommunitasStaking.sol#170)
is not in mixedCase
Parameter KommunitasStaking.getTotalLockedTokens(address)._of (KommunitasStaking.sol#185) is not
 in mixedCase
Parameter KommunitasStaking.getUserPendingRewards(address)._of (KommunitasStaking.sol#201) is no
t in mixedCase
Parameter KommunitasStaking.getUserStakedTokens(address)._of (KommunitasStaking.sol#216) is not
in mixedCase
Parameter KommunitasStaking.setDurationMultiplier(uint256,uint256)._multiplier (KommunitasStakin
g.sol#234) is not in mixedCase
Parameter KommunitasStaking.setDurationMultiplier(uint256,uint256)._duration (KommunitasStaking.
sol#234) is not in mixedCase
Parameter KommunitasStaking.updateLockDuration(uint256,uint256)._minDuration (KommunitasStaking.
sol#244) is not in mixedCase
Parameter KommunitasStaking.updateLockDuration(uint256,uint256)._maxDuration (KommunitasStaking.
sol#244) is not in mixedCase
Parameter KommunitasStaking.updatePenaltyFees(uint256)._feesPercentage (KommunitasStaking.sol#25
0) is not in mixedCase
Parameter KommunitasStaking.updateAPY(uint256)._apy (KommunitasStaking.sol#256) is not in mixedC
ase
Constant KommunitasStaking.yearDuration (KommunitasStaking.sol#23) is not in UPPER_CASE_WITH_UND
ERSCORES
Parameter KommunitasVoting.mint(address,uint256)._to (KommunitasVoting.sol#16) is not in mixedCa
se
Parameter KommunitasVoting.mint(address,uint256)._amount (KommunitasVoting.sol#16) is not in mix
edCase
Parameter KommunitasVoting.burn(address,uint256)._to (KommunitasVoting.sol#21) is not in mixedCa
se
Parameter KommunitasVoting.burn(address,uint256)._amount (KommunitasVoting.sol#21) is not in mix
edCase
Variable KommunitasVoting._delegates (KommunitasVoting.sol#27) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity
-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (../OpenZeppelin/access/Ownable.sol#53-55)
```

```
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (../OpenZeppelin/access/Ownable.sol#61-64)
symbol() should be declared external:
        - ERC20.symbol() (../OpenZeppelin/token/ERC20/ERC20.sol#69-71)
decimals() should be declared external:
        - ERC20.decimals() (../OpenZeppelin/token/ERC20/ERC20.sol#86-88)
totalSupply() should be declared external:
        - ERC20.totalSupply() (../OpenZeppelin/token/ERC20/ERC20.sol#93-95)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (../OpenZeppelin/token/ERC20/ERC20.sol#112-115)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (../OpenZeppelin/token/ERC20/ERC20.sol#131-134)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (../OpenZeppelin/token/ERC20/ERC20.sol#149
-163)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (../OpenZeppelin/token/ERC20/ERC20.sol#177-18
0)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (../OpenZeppelin/token/ERC20/ERC20.sol#196-20
4)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (../OpenZeppelin/token/ERC20/extensions/ERC20Burnable.sol#
19-21)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (../OpenZeppelin/token/ERC20/extensions/ERC20B
urnable.sol#34-41)
getTotalWithdrawableTokens(address) should be declared external:
        - KommunitasStaking.getTotalWithdrawableTokens(address) (KommunitasStaking.sol#170-179)
getTotalLockedTokens(address) should be declared external:
        - KommunitasStaking.getTotalLockedTokens(address) (KommunitasStaking.sol#185-194)
getUserPendingRewards(address) should be declared external:
        - KommunitasStaking.getUserPendingRewards(address) (KommunitasStaking.sol#201-210)
mint(address,uint256) should be declared external:
        - KommunitasVoting.mint(address,uint256) (KommunitasVoting.sol#16-19)
burn(address,uint256) should be declared external:
        - KommunitasVoting.burn(address,uint256) (KommunitasVoting.sol#21-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-co
uld-be-declared-external
```

```
Reentrancy in KommunitasStaking.preMatureWithdraw() (KommunitasStaking.sol#132-164):
        External calls:
        - komToken.burn(penaltyAmount) (KommunitasStaking.sol#153)
        - komToken.transfer(msg.sender,withdrawableAmount) (KommunitasStaking.sol#156)
        - komvToken.burn(msg.sender,1) (KommunitasStaking.sol#159)
        Event emitted after the call(s):
        - EmergencyUnlocked(msg.sender,unlockableTokens) (KommunitasStaking.sol#161)
Reentrancy in KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127):
        External calls:
        - komToken.transfer(_of,unlockableTokens) (KommunitasStaking.sol#119)
        - komvToken.burn(_of,1) (KommunitasStaking.sol#122)
        Event emitted after the call(s):
        - Unlocked(_of,unlockableTokens) (KommunitasStaking.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilit
ies-3
INFO:Detectors:
KommunitasStaking.unlock(address) (KommunitasStaking.sol#106-127) uses timestamp for comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed (KommunitasStakin
g.sol#111)
KommunitasStaking.getTotalWithdrawableTokens(address) (KommunitasStaking.sol#170-179) uses times
tamp for comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed (KommunitasStakin
g.sol#174)
KommunitasStaking.getTotalLockedTokens(address) (KommunitasStaking.sol#185-194) uses timestamp f
or comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity > block.timestamp && ! locks[_of][i].claimed (KommunitasStaking
.sol#189)
KommunitasStaking.getUserPendingRewards(address) (KommunitasStaking.sol#201-210) uses timestamp
for comparisons
        Dangerous comparisons:
        - locks[_of][i].maturity <= block.timestamp && ! locks[_of][i].claimed (KommunitasStakin
g.sol#205)
KommunitasVoting.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (KommunitasVoting.
sol#85-126) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,KOMV::delegateBySig: signature expired)
 (KommunitasVoting.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (../OpenZeppelin/utils/Address.sol#26-36) uses assembly
```

## Conclusion:

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

# 4. Conclusion

We examined the design and implementation of Kommunitas in this audit. The present code base is well-organized. We would much appreciate any constructive input or ideas regarding our methodology, audit findings, or potential scope/coverage gaps in this report.


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