Kommunitas Staking V3

Smart Contract Security Audit

Prepared by ShellBoxes

Jan 2nd, 2023 - Jan 7th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Kommunitas
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/Kommunitas-net/staking-v3	f6bdf3df8ee71645e8863bc394bc18dfe57d4b6f

Re-Audit

Repository	Commit Hash
https://github.com/Kommunitas-net/staking-v3	ab2aadb8fc93a743df2e01a838e6ef32d8712be2

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Kommunitas engaged ShellBoxes to conduct a security assessment on the Kommunitas Staking V3 beginning on Jan 2nd, 2023 and ending Jan 7th, 2023. 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	Solidity Smart Contract
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 Staking V3 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 , 3 high-severity, 4 medium-severity, 5 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. A User Can Get More Than A Single KomV Token	HIGH	Fixed
SHB.2. Any Worker Can Unstake to Any User Staker	HIGH	Mitigated
SHB.3. Savior has Unrestricted Power to Withdraw Tokens with emergencyWithdraw Function	HIGH	Acknowledged
SHB.4. Owner Can Change Any Compound Type	MEDIUM	Fixed
SHB.5. Lack of Two-Factor Verification for Updating Admin Proxy Address	MEDIUM	Fixed
SHB.6. Centralization Risk	MEDIUM	Acknowledged
SHB.7. Owner Can Renounce Ownership	MEDIUM	Fixed
SHB.8. Race Condition	LOW	Fixed
SHB.9. Missing Token Address Verification	LOW	Fixed
SHB.10. Missing Value Verification	LOW	Fixed

SHB.11. Missing Address Verification	LOW	Fixed
SHB.12. Disynchronization between the workerNumber and the actual number of workers	LOW	Fixed

3 Finding Details

SHB.1 A User Can Get More Than A Single KomV Token

- Severity : HIGH
- Status : Fixed
- Likelihood : 3
- Impact : 2

Description:

The KomV token is an ERC20 which is minted to a staker if their staked amount has reached the minGetKomV value. It is used for voting, which means a user should not be able to vote more than once, this implies that the user should not have more than 1 token. However, the contract's logic allows a user to get more than a single voting token minted.This issue is caused by the contract checking the user's current balance, instead of verifying if they have already been minted a voting token.

Exploit Scenario:

1. The attacker stakes minGetKomV amount.
2. The _stake function checks if the attacker already has a KomV token by checking their balance.
3. Since the attacker's balance is 0, a KomV token is minted and given to the attacker.
4. The attacker transfers the KomV token to an account they own.
5. The attacker unstakes the whole amount.
6. The _unstake function checks if the attacker still has the KomV token by checking their balance.
7. Since the balance is 0 (because the KomV token was transferred to another account.)
8. The attacker repeats this process as many times as they want to obtain more KomV tokens.

Files Affected:

SHB.1.1: KommunitasStakingV3.sol

```
if(
    stakerDetail[_sender].stakedAmount >= minGetKomV &&
    IERC20MintableBurnableUpgradeable(komVToken).balanceOf(_sender) == 0
  ){
    IERC20MintableBurnableUpgradeable(komVToken).mint(_sender, 1);
   }
```

SHB.1.2: KommunitasStakingV3.sol

```
if(
    stakerDetail[_sender].stakedAmount < minGetKomV &&
    IERC20MintableBurnableUpgradeable(komVToken).balanceOf(_sender) > 0
    ){
    IERC20MintableBurnableUpgradeable(komVToken).burn(_sender, 1);
    }
```

Recommendation:

Consider using a mapping of users who own the KomV token so that a staker should be the only one owning and responsible for that KomV token. The mapping will look as follows : mapping(address => bool) public hasKomV;

Updates

The Kommunitas team resolved the issue by adding the hasKomV mapping to identify the users who already minted their KomV token.

SHB.1.3: KommunitasStakingV3.sol

```
    if(
      stakerDetail[_sender].stakedAmount >= minGetKomV &&
      IERC20MintableBurnableUpgradeable(komVToken).balanceOf(_sender) ==
          ↪ 0 &&
      !hasKomV[_sender]
    ){
      IERC20MintableBurnableUpgradeable(komVToken).mint(_sender, 1);
      hasKomV[_sender] = true;
    }
```

SHB.1.4: KommunitasStakingV3.sol

```
      stakerDetail[_sender].stakedAmount < minGetKomV &&
      IERC20MintableBurnableUpgradeable(komVToken).balanceOf(_sender) >
          ↪ 0 &&
      hasKomV[_sender]
    ){
      IERC20MintableBurnableUpgradeable(komVToken).burn(_sender, 1);
      hasKomV[_sender] = false;
    }
```

SHB.2 Any Worker Can Unstake to Any User Staker

- Severity : HIGH
- Status : Mitigated
- Likelihood : 2
- Impact : 3

Description:

The contract's unstake function allows any worker to unstake tokens from any staker, regardless of whether the worker has a permission or authorization to do so. This can potentially allow a worker to unstake tokens from a staker without their consent or knowledge, potentially leading to loss of funds or other negative consequences.

Files Affected:

SHB.2.1: KommunitasStakingV3.sol

```
function unstake(
    uint232 _userStakedIndex,
    uint256 _amount,
    address _staker
  ) external virtual whenNotPaused {
    // set staker
    if(!isWorker[_msgSender()]) _staker = _msgSender();
```

Recommendation:

To ensure that only the caller is able to unstake their own tokens in the unstake function, you can modify the function as follows:

SHB.2.2: KommunitasStakingV3.sol

```
function unstake(uint232 _userStakedIndex, uint256 _amount) external
   ↪ virtual whenNotPaused {
  address _staker = _msgSender();
  .....
}
```

This ensures that only the caller is able to unstake their own tokens, rather than allowing a worker to unstake tokens on behalf of someone else.

Updates

The Kommunitas team mitigated the risk by preventing the worker from unstaking before the due date.

SHB.2.3: KommunitasStakingV3.sol

```
  function unstake(
    uint232 _userStakedIndex,
    uint256 _amount,
    address _staker
  ) external virtual whenNotPaused {
    // get stake data
    Stake memory stakeDetail = staked[_staker][_userStakedIndex];

    // worker check
    if(isWorker[_msgSender()]){
      require(block.timestamp > stakeDetail.endedAt, "premature");
    } else {
      _staker = _msgSender();
    }
```

SHB.3 Savior has Unrestricted Power to Withdraw Tokens with emergencyWithdraw Function

- Severity : HIGH
- Status : Acknowledged
- Likelihood : 2
- Impact : 3

Description:

The emergencyWithdraw function in the contract allows the savior address to withdraw any amount of tokens from the contract, without any restrictions or limitations. This gives the savior address excessive power and could potentially be abused.

Exploit Scenario:

The savior address could potentially use the emergencyWithdraw function to withdraw a large amount of tokens from the contract, potentially causing financial harm to the contract or its users. Also, the savior can set his address as a receiver and will be able to get the tokens.

Files Affected:

SHB.3.1: KommunitasStakingV3.sol

```
function emergencyWithdraw(
    address _token,
    uint256 _amount,
    address _receiver
  ) external virtual {
    onlySavior();

    // adjust amount to wd
    uint256 balance = IERC20Upgradeable(_token).balanceOf(address(this));
    if(_amount > balance) _amount = balance;

    IERC20MintableBurnableUpgradeable(_token).safeTransfer(
      _receiver,
      _amount
    );
  }
}
```

Recommendation:

It is recommended to implement restrictions or limitations on the emergencyWithdraw function to prevent the savior address from having unrestricted power to withdraw tokens from the contract. This could include implementing a maximum withdrawal limit or requiring additional approvals or checks before allowing the savior address to withdraw tokens.

Updates

The Kommunitas team acknowledged the risk, stating that the emergencyWithdraw is a part of the business logic for safety, and they are utilizing a multisig contract as a savior, which needs 2-of-3 approvals to perform the transaction.

SHB.4 Owner Can Change Any Compound Type

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The contract's changeCompoundType function allows the owner to change the compound type of any staked tokens, regardless of whether they have permission or authorization to do so. This can potentially allow the owner to change the compound type of user's staked tokens without their consent, potentially leading to unexpected or unintended consequences for the affected user.

Files Affected:

SHB.4.1: KommunitasStakingV3.sol

```
function changeCompoundType(
  address _staker,
  uint232 _userStakedIndex,
  CompoundTypes _newCompoundType
) external virtual whenNotPaused {
  // owner validation
  if(_msgSender() != owner()) _staker = _msgSender();
```

Recommendation:

Consider removing the owner's power to change the compound type for a staker and set the _staker variable to the caller's address (msg.sender). This ensures that only the caller is able to change the compound type for their stake, rather than the owner having the power to do so for any staker.

Updates

The Kommunitas team resolved the issue by preventing the owner from changing the compound type for the users.

SHB.5 Lack of Two-Factor Verification for Updating Admin Proxy Address

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The transferProxyAdmin function, in the AdminProxyManager contract, allows the current proxyAdmin() to update the admin proxy address without any additional verification or authentication. This can lead to permanently giving the admin role to a wrong admin, which cannot be revoked again.

Exploit Scenario:

When this function is called with a mistaken address as parameter by the existing adminProxy, the AdminProxyManager privileges are immediately transferred to this unknown address.The original admin will lose the contract and will be unable to retrieve their control.

Files Affected:

SHB.5.1: AdminProxyManager.sol

```
function transferProxyAdmin(address _newProxyAdmin) external virtual
   ↪ proxied {
  require(_newProxyAdmin != address(0) && _newProxyAdmin !=
     ↪ _proxyAdmin(), "bad");
```

```

    assembly {
      sstore(0
          ↪ xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b
      5d6103, _newProxyAdmin)
    }
```

Recommendation:

Consider adding an extra function to permit the newly assigned admin to claim proxyAdmin control. This will stop the issue of automatic assignment of a mistaken address. When an address is set as the admin using the transferProxyAdmin function, the contract can have an additional function.For instance,updateProxyAdmin will be called by the assigned person.The later becomes the new ProxyAdmin and then the former admin no longer has the proxy admin privileges.

Updates

The Kommunitas team resolved the issue by implementing two-factor verification, this was done by adding the _pendingProxyAdmin variable and the acceptProxyAdmin function that allows the new admins to claim their ownership.

SHB.5.2: AdminProxyManager.sol

```
  function acceptProxyAdmin() external {
    address sender = msg.sender;
    require(_pendingProxyAdmin == msg.sender, "bad");
    _transferProxyAdmin(sender);
  }
```

SHB.6 Centralization Risk

- Severity : MEDIUM
- Status : Acknowledged

- Likelihood : 1
- Impact : 3

Description:

The functions changeCompoundType, setMin, setPeriodInDays, setPenaltyFee and setAPY are functions that modify values related to each lock index. However, the owner is the only one able to change these values at any time, without the consent of the stakers.

Exploit Scenario:

1. An owner can set the minStaking value to a low value and set the APY to a high value to motivate users to lock their funds.
2. The owner has enough stakers. The owner immediately sets the penaltyFee to a really high value, and changes all the staker's compoundTypes, and the APY to 0. Hence, the stakers are unable to unstake their funds.

Files Affected:

SHB.6.1: KommunitasStakingV3.sol

```
function changeCompoundType(
    address _staker,
    uint232 _userStakedIndex,
    CompoundTypes _newCompoundType
  ) external virtual whenNotPaused {
    // owner validation
    if(_msgSender() != owner()) _staker = _msgSender();

    // get stake data
    Stake memory stakeDetail = staked[_staker][_userStakedIndex];

    require(
      staked[_staker].length > _userStakedIndex && // user staked index validation
      stakeDetail.compoundType != _newCompoundType, // compound type validation
      "bad"
    );

    // assign new compound type
    staked[_staker][_userStakedIndex].compoundType = _newCompoundType;
  }
```

SHB.6.2: KommunitasStakingV3.sol

```
function setMin(
    uint64 _minStaking,
    uint64 _minPrivatePartner,
    uint64 _minGetKomV,
    uint16 _minLockIndexGetGiveaway
  ) external virtual onlyOwner {
    if(_minStaking > 0) minStaking = _minStaking;
    if(_minPrivatePartner > 0){
      minPrivatePartner = _minPrivatePartner;
      privatePartnerStakedAmount = 0; // reset private partner total staked amount
    }
    if(_minGetKomV > 0) minGetKomV = _minGetKomV;
    if(_minLockIndexGetGiveaway > 0){
      minLockIndexGetGiveaway = _minLockIndexGetGiveaway;
      giveawayStakedAmount = 0; // reset giveaway total staked amount
    }
  }
```

SHB.6.3: KommunitasStakingV3.sol

```
function setPeriodInDays(
    uint16 _lockIndex,
    uint128 _newLockPeriodInDays
  ) external virtual onlyOwner {
    require(lockNumber > _lockIndex, "bad");
    lock[_lockIndex].lockPeriodInSeconds = _newLockPeriodInDays * 86400;
  }
```

SHB.6.4: KommunitasStakingV3.sol

```
function setPenaltyFee(
    uint16 _lockIndex,
    uint64 _feeInPercent_d2
  ) external virtual onlyOwner {
    require(lockNumber > _lockIndex, "bad");
    lock[_lockIndex].feeInPercent_d2 = _feeInPercent_d2;
  }
```

SHB.6.5: KommunitasStakingV3.sol

```
function setAPY(
    uint16 _lockIndex,
    uint64 _apy_d2
  ) external virtual onlyOwner {
    require(lockNumber > _lockIndex, "bad");
    lock[_lockIndex].apy_d2 = _apy_d2;
  }
```

SHB.6.6: KommunitasStakingV3.sol

```
function togglePause() external onlyOwner virtual {
    if(paused()){
      _unpause();
    } else {
      _pause();
    }
  }
```

Recommendation:

Since these functions modify state variables related to the stakers, such changes should be proposed to the stakers, and the majority should either accept or deny these proposals.

Updates

The Kommunitas team acknowledged the risk, stating that they are using a multisig wallet, and they are planning to implement a governance system to enable stakers to vote on new proposals.

SHB.7 Owner Can Renounce Ownership

- Severity : MEDIUM
- Status : Fixed

- Likelihood : 1
- Impact : 3

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Files Affected:

SHB.7.1: KommunitasStakingV3.sol

```
contract KommunitasStakingV3 is
  Initializable,
  UUPSUpgradeable,
  OwnableUpgradeable ,
  PausableUpgradeable,
  AdminProxyManager,
IKommunitasStakingV3
```

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Updates

The Kommunitas team resolved the issue by removing the renounceOwnership function from the OwnableUpgradeable contract.

SHB.8 Race Condition

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The setMin function in the contract allows the owner to update the minStaking variable, which is used to validate the minimum amount required for staking in the stake function. However, the setMin function does not use any synchronization mechanism to prevent concurrent access, which could lead to a race condition. This could lead to unpredictable behavior in the stake function, since the minStaking variable may be updated concurrently with a stake transaction.

Files Affected:

SHB.8.1: KommunitasStakingV3.sol

```
 function setMin(
  uint64 _minStaking,
  uint64 _minPrivatePartner,
  uint64 _minGetKomV,
  uint16 _minLockIndexGetGiveaway
) external virtual onlyOwner {
  if(_minStaking > 0) minStaking = _minStaking;
  if(_minPrivatePartner > 0){
    minPrivatePartner = _minPrivatePartner;
    privatePartnerStakedAmount = 0; // reset private partner total
        ↪ staked amount
  }
  if(_minGetKomV > 0) minGetKomV = _minGetKomV;
  if(_minLockIndexGetGiveaway > 0){
    minLockIndexGetGiveaway = _minLockIndexGetGiveaway;
    giveawayStakedAmount = 0; // reset giveaway total staked amount
  }
}
```

Recommendation:

To fix the race condition issue in the setMin function, you can use a synchronization mechanism such as a require statement to ensure that the minStaking variable is not updated concurrently with a stake transaction.

Updates

The Kommunitas team resolved the issue by requiring the contract to be paused before executing the setMin function.

SHB.8.2: KommunitasStakingV3.sol

```
function setMin(
    uint64 _minStaking,
    uint64 _minPrivatePartner,
    uint64 _minGetKomV,
    uint16 _minLockIndexGetGiveaway
  ) external virtual whenPaused onlyOwner {
    if(_minStaking > 0) minStaking = _minStaking;
    if(_minPrivatePartner > 0){
      minPrivatePartner = _minPrivatePartner;
      privatePartnerStakedAmount = 0; // reset private partner total staked amount
    }
    if(_minGetKomV > 0) minGetKomV = _minGetKomV;
    if(_minLockIndexGetGiveaway > 0){
      minLockIndexGetGiveaway = _minLockIndexGetGiveaway;
      giveawayStakedAmount = 0; // reset giveaway total staked amount
    }

    // unpause
    _unpause();
  }
```

SHB.9 Missing Token Address Verification

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

The contract's init function allows the setting of komToken and komVToken token addresses without verifying that the addresses are contract addresses. This can potentially allow an attacker to set these token addresses to non-contract addresses or address(0).

Files Affected:

SHB.9.1: KommunitasStakingV3.sol

```
komToken = _komToken;
   komVToken = _komVToken;
```

Recommendation:

It is recommended to verify that the addresses being set as the komToken and komVToken token addresses are indeed contract addresses. This can be done by calling the isContract function on the addresses in question. This function is provided by the Ethereum Contract Address Validation library, which can be found here: Address.sol

Updates

The Kommunitas team resolved the issue by using the isContract function to make sure the _komToken and the _komVToken addresses refer to smart contracts.

SHB.9.2: KommunitasStakingV3.sol

```
    require(
      _lockPeriodInDays.length == _apy_d2.length &&
      _lockPeriodInDays.length == _feeInPercent_d2.length &&
      AddressUpgradeable.isContract(_komToken) &&
      AddressUpgradeable.isContract(_komVToken) &&
      _savior != address(0),
      "misslength"
    );
```

SHB.10 Missing Value Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

There are three functions setPeriodInDays, setPenaltyFee, and setAPY that are used to update state variables related to the staking process. These functions can only be accessed by the owner. However, there are no checks in place to ensure that the values of these state variables are not set to unreasonable values. For example, there is no check to prevent the APY from being set to 0 or the penaltyFee from being set to a large number.

Files Affected:

SHB.10.1: KommunitasStakingV3.sol

```
function setPeriodInDays(
    uint16 _lockIndex,
    uint128 _newLockPeriodInDays
  ) external virtual onlyOwner {
    require(lockNumber > _lockIndex, "bad");
    lock[_lockIndex].lockPeriodInSeconds = _newLockPeriodInDays * 86400;
  }
```

SHB.10.2: KommunitasStakingV3.sol

```
function setPenaltyFee(
    uint16 _lockIndex,
    uint64 _feeInPercent_d2
  ) external virtual onlyOwner {
    require(lockNumber > _lockIndex, "bad");
    lock[_lockIndex].feeInPercent_d2 = _feeInPercent_d2;
  }
```

SHB.10.3: KommunitasStakingV3.sol

```
function setAPY(
    uint16 _lockIndex,
    uint64 _apy_d2
  ) external virtual onlyOwner {
    require(lockNumber > _lockIndex, "bad");
    lock[_lockIndex].apy_d2 = _apy_d2;
  }
```

Recommendation:

For the setPeriodInDays function, it is recommended to use a list of predefined options (such as 30, 90, 120,..) rather than allowing the input of any number.
For the setPenaltyFee function, it is recommended to set a maximum limit for the penalty fee that cannot be exceeded. Similarly, for the setAPY function, it is recommended to set a minimum value that must be met.

Updates

The Kommunitas team mitigated the risk by verifying the arguments of the setPeriodInDays and the setPenaltyFee, and verifying the upper limit in the setAPY function.

SHB.10.4: KommunitasStakingV3.sol

```
  function setPeriodInDays(
    uint16 _lockIndex,
    uint128 _newLockPeriodInDays
  ) external virtual onlyOwner {
    require(
      lockNumber > _lockIndex &&
      _newLockPeriodInDays >= 86400 &&
      _newLockPeriodInDays <= (5 * yearInSeconds),
      "bad"
    );
    lock[_lockIndex].lockPeriodInSeconds = _newLockPeriodInDays * 86400;
  }
```

SHB.10.5: KommunitasStakingV3.sol

```
  function setPenaltyFee(
    uint16 _lockIndex,
    uint64 _feeInPercent_d2
  ) external virtual onlyOwner {
```

```
  require(
    lockNumber > _lockIndex &&
    _feeInPercent_d2 >= 100 &&
    _feeInPercent_d2 < 10000,
    "bad"
  );
  lock[_lockIndex].feeInPercent_d2 = _feeInPercent_d2;
}
```

SHB.10.6: KommunitasStakingV3.sol

```
function setAPY(
  uint16 _lockIndex,
  uint64 _apy_d2
) external virtual onlyOwner {
  require(
    lockNumber > _lockIndex &&
    _apy_d2 < 10000,
    "bad"
  );
  lock[_lockIndex].apy_d2 = _apy_d2;
}
```

SHB.11 Missing Address Verification

- Severity : LOW
- Status : Fixed

- Likelihood : 1
- Impact : 2

Description:

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

Files Affected:

SHB.11.1: KommunitasStakingV3.sol

```solidity
function setSavior(address _savior) external virtual {
    onlySavior();
    savior = _savior;
  }
```

SHB.11.2: KommunitasStakingV3.sol

```solidity
function init(
    address _komToken,
    address _komVToken,
    uint128[] calldata _lockPeriodInDays,
    uint64[] calldata _apy_d2,
    uint64[] calldata _feeInPercent_d2,
    address _savior
  ) external initializer proxied {
    __UUPSUpgradeable_init();
    __Pausable_init();
    __Ownable_init();
    __AdminProxyManager_init(_msgSender());

    require(
      _lockPeriodInDays.length == _apy_d2.length &&
      _lockPeriodInDays.length == _feeInPercent_d2.length,
      "misslength"
    );

    komToken = _komToken;
    komVToken = _komVToken;
    lockNumber = uint16(_lockPeriodInDays.length);
    savior = _savior;
```

SHB.11.3: KommunitasStakingV3.sol

```
function addWorker(address _worker) external virtual onlyOwner {
    isWorker[_worker] = true;
    ++workerNumber;
  }
```

SHB.11.4: KommunitasStakingV3.sol

```
function removeWorker(address _worker) external virtual onlyOwner {
    isWorker[_worker] = false;
    --workerNumber;
  }
```

SHB.11.5: KommunitasStakingV3.sol

```
function changeWorker(
    address _oldWorker,
    address _newWorker
  ) external virtual onlyOwner {
    isWorker[_oldWorker] = false;
    isWorker[_newWorker] = true;
  }
```

SHB.11.6: KommunitasStakingV3.sol

```
function toggleTrustedForwarder(address _forwarder) external virtual
    ↪ onlyOwner {
    isTrustedForwarder[_forwarder] = !isTrustedForwarder[_forwarder];
  }
```

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Updates

The Kommunitas team resolved the issue by verifying all the address arguments to be different from the address(0).

SHB.12 Disynchronization between the workerNumber and the actual number of workers

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 1

Description:

The addWorker and removeWorker functions, add the worker's address to a mapping and then increment or decrement the workerNumber variable to keep track of the number of workers currently active in the contract. However, these functions do not check if a worker already exists(in case of addWorker) or not (in case of removeWorker), and still increments the workerNumber which causes a mismatch between the workerNumber and the actual number of workers.

Files Affected:

SHB.12.1: KommunitasStakingV3.sol

```
function addWorker(address _worker) external virtual onlyOwner {
   isWorker[_worker] = true;
   ++workerNumber;
 }
```

SHB.12.2: KommunitasStakingV3.sol

```
function removeWorker(address _worker) external virtual onlyOwner {
    isWorker[_worker] = false;
    --workerNumber;
```

```
 }
```

Recommendation:

Consider adding a require statement that checks that the entered address is not address(0) and that it does not already exist in the addWorker. This can be done as follows :

SHB.12.3: KommunitasStakingV3.sol

```
function addWorker(address _worker) external virtual onlyOwner {
   require(_worker != address(0) && !isWorker[_worker], "worker already exists");
   isWorker[_worker] = true;
   ++workerNumber;
 }
```

and for the removeWorker the code will look something like this :

SHB.12.4: KommunitasStakingV3.sol

```
function removeWorker(address _worker) external virtual onlyOwner {
   require(isWorker[_worker], "worker does not exist");
   isWorker[_worker] = false;
   --workerNumber;
 }
```

Updates

The Kommunitas team resolved the issue by preventing the addition of the address(0) and the modification of the workerNumber if the address already exists.

4 Best Practices

BP.1 Using a Solidity Modifier to Encapsulate onlySavior Checks

Description:

It is generally a good practice to use Solidity modifiers to encapsulate and reuse common checks or functionality in a contract. Modifiers allow you to define a set of conditions or requirements that must be met in order to execute the code in a function or method. In the case of the onlySavior function, it appears to be used to enforce that only the savior address is allowed to execute certain functions or methods in the contract. However, using a separate function to perform this check can be somewhat inefficient and can potentially lead to code duplication if the same check is needed in multiple functions.

1. Define a modifier named onlySavior that contains the check for the savior address. For example:

BP.1.1: KommunitasStakingV3.sol

```
modifier onlySavior() {
    require(_msgSender() == savior, "!savior");
    _;
}
```

2. Apply the onlySavior modifier to any functions or methods that should only be accessible to the savior address. For example:

BP.1.2: KommunitasStakingV3.sol

```
function someFunction() public onlySavior {
    // Function body
}
```

Status - Not Fixed

BP.2 Optimizing Code Quality and Readability with Separate Pause/Unpause Functions

Description:

In the togglePause function, the pause/unpause logic is currently encapsulated in a single function. This can potentially make the code more difficult to read and understand, as the purpose and behavior of the function may not be immediately clear. One potential solution to improve the readability and clarity of this code is to separate the pause/unpause logic into two separate functions. For example:

BP.2.1: KommunitasStakingV3.sol

```
function pause() external onlyOwner virtual {
    _pause();
}

function unpause() external onlyOwner virtual {
    _unpause();
}
```

This approach allows you to clearly distinguish the pause and unpause functionality and make it more explicit in the contract code. It also allows you to give the functions descriptive names that reflect their purpose, which can make the code easier to understand and maintain.

Status - Not Fixed

BP.3 Optimize Event Emission by Combining Functions

Description:

The contract includes the emitUnstaked function that only emits an event and does not perform any other actions. This can potentially lead to unnecessary gas costs and code complexity. It is recommended to optimize the contract by removing the function that only emits an event ,and adding the event emission directly to the core function that performs additional actions. This can help reduce gas costs and code complexity by reducing the number of function calls and events that are emitted.

Files Affected:

BP.3.1: KommunitasStakingV3.sol

```
function emitUnstaked(
  address _stakerAddress,
  uint128 _lockPeriodInDays,
  CompoundTypes _compoundType,
  uint256 _amount,
  uint256 _reward,
  uint256 _penaltyPremature,
  uint128 _stakedAt,
  uint128 _endedAt,
  bool _isPremature
 ) internal virtual {
  emit Unstaked(
    _stakerAddress,
    _lockPeriodInDays,
    _compoundType,
    _amount,
    _reward,
      _penaltyPremature,
      _stakedAt,
      _endedAt,
      uint128(block.timestamp),
      _isPremature
    );
  }
```

Status - Not Fixed

5 Tests

→ StakingV3 (27 passing (1m))

✓ Success: Stake 100 kom in no compounding type (415ms)

✓ Success: Unstake 100 kom in no compounding type (46ms)

✓ Success: Stake 100 kom in rewardOnly compounding type

✓ Success: Unstake 100 kom in rewardOnly compounding type (65ms)

✓ Success: Stake 100 kom in principalAndReward compounding type

✓ Success: Unstake 100 kom in principalAndReward compounding type (53ms)

✓ Success: Full premature unstake 100 kom in no compounding type (64ms)

✓ Success: Partial premature unstake 80 of 100 kom in no compounding type (68ms)

✓ Success: Full premature unstake 100 kom in rewardOnly compounding type (54ms)

✓ Success: Partial premature unstake 80 of 100 kom in rewardOnly compounding type (67ms)

✓ Success: Full premature unstake 100 kom in principalAndReward compounding type (54ms)

✓ Success: Partial premature unstake 80 of 100 kom in principalAndReward compounding type (67ms)

✓ Success: Stake 500k kom in no compounding type

- ✓ Success: Unstake 500k kom in no compounding type (59ms)
- ✓ Success: Stake 500k kom in rewardOnly compounding type
- ✓ Success: Unstake 500k kom in rewardOnly compounding type (65ms)
- ✓ Success: Stake 500k kom in principalAndReward compounding type
- ✓ Success: Unstake 500k kom in principalAndReward compounding type (62ms)
- ✓ Success: Full premature unstake 500k kom in no compounding type (64ms)
- ✓ Success: Partial premature unstake 600k of 800k kom in no compounding type (76ms)
- ✓ Success: Partial premature unstake 200k of 800k kom in no compounding type (76ms)
- ✓ Success: Full premature unstake 500k kom in rewardOnly compounding type (64ms)
- ✓ Success: Partial premature unstake 600k of 800k kom in rewardOnly compounding type (75ms)
- ✓ Success: Partial premature unstake 200k of 800k kom in rewardOnly compounding type (74ms)
- ✓ Success: Full premature unstake 500k kom in principalAndReward compounding type (54ms)
- ✓ Success: Partial premature unstake 600k of 800k kom in principalAndReward compounding type (74ms)

```
✓ Success: Partial premature unstake 200k of 800k kom in principalAndReward compounding type (73ms)
```

6 Conclusion

In this audit, we examined the design and implementation of Kommunitas Staking V3 contract and discovered several issues of varying severity. Kommunitas team addressed 9 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.

7 Scope Files

7.1 Audit

Files	MD5 Hash
contracts/KommunitasStakingV3.sol	a4a94d6910cce3457860ebac89e70fd0
contracts/interface/IERC20MintableBurnableUpgradeable.sol	9af652f839f640e7a7884ae356963f18
contracts/interface/IKommunitasStakingV3.sol	e6571d83f8da80268f88c72078f3f084
contracts/util/AdminProxyManager.sol	4b03425e63129be5e9c3c3744e760370
contracts/util/ERC1967.sol	51185e23ee344363c77388be75e0c0e1

7.2 Re-Audit

Files	MD5 Hash
contracts/KommunitasStakingV3.sol	fcdeac0b5c31867867ee004381db8d39
contracts/util/AdminProxyManager.sol	5faab490d0f406375288c3ed8d7068c5
contracts/util/OwnableUpgradeable.sol	919731340efa4bb64950ec96f37051e7
contracts/interface/IERC20MintableBurnableUpgradeable.sol	9af652f839f640e7a7884ae356963f18
contracts/interface/IKommunitasStakingV3.sol	e6571d83f8da80268f88c72078f3f084
contracts/util/ERC1967.sol	51185e23ee344363c77388be75e0c0e1

8 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology’s proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don’t offer any kind of investing advice and shouldn’t be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

SHELLBOXES

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