Kommunitas Token

Smart Contract Security Audit

Prepared by ShellBoxes

June 3rd, 2025 - June 7th, 2025

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Kommunitas
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/Kommunitas-net/core-contract	2459b7d1ed02f4249be29bdb80343653d45a8792

Re-Audit

Repository	Commit Hash
https://github.com/Kommunitas-net/core-contract	fc4fbdd1c4a5ad7eae597987509368431ac6ad30

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Kommunitas engaged ShellBoxes to conduct a security assessment on the Kommunitas Token beginning on June 3rd, 2025 and ending June 7th, 2025. 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. Kommunitas is the solution for Multi Chain oriented projects. Kommunitas welcomes project from various blockchain like Polygon, BSC, Ethereum, Avalance, Solana, etc...

Issuer	Kommunitas
Website	`https://www.kommunitas.net`
Type	Solidity Smart Contract
Documentation	Kommunitas Docs
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	Likelihood: High	Likelihood: Medium	Likelihood: Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Kommunitas Token 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, 4 high-severity, 1 medium-severity, 4 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Wrong maxSupply maths caps supply at only 1600 tokens	CRITICAL	Fixed
SHB.2. mint() is restricted to the admin of MINTER_ROLE, not to the minter itself	CRITICAL	Fixed
SHB.3. Empty `_authorizeUpgrade()` makes upgrade safety depend on an external modifier	HIGH	Acknowledged
SHB.4. Hard-revert on transfers to the token contract breaks integrations and loses funds	HIGH	Fixed
SHB.5. `pause()` does not stop minting / burning	HIGH	Fixed
SHB.6. `setMaxSupply()` can raise the cap to an arbitrary value	HIGH	Fixed

SHB.8. No storage-gap reserved for future upgrades	MEDIUM	Fixed
SHB.7. `KommunitasTokenSelfTransferred` uses the spender instead of the token-owner	LOW	Fixed
SHB.9. No way to rescue accidentally sent ERC-20 or Ether	LOW	Fixed
SHB.10. `ERC20Permit` cached domain-separator breaks on chain-ID fork	LOW	Mitigated
SHB.11. Self-transfer guard can be bypassed via `increaseAllowance()`	LOW	Fixed

3 Finding Details

SHB.1 Wrong maxSupply maths caps supply at only 1600 tokens

- Severity: CRITICAL
- Status: Fixed
- Likelihood: 3
- Impact: 3

Description:

init() tries to fall back to 2 billion tokens with 8 decimals, yet it multiplies instead of exponentiating.2 × 10**9 × 10 × 8 = 160 000 000 000 base-units which is 1600 whole tokens. Once that amount is minted every further mint() reverts, freezing any dependent protocol.

Files Affected:

SHB.1.1: KommunitasToken.sol

```
50  if (maxSupply_ == 0) maxSupply_ = 2 * 1e9 * 10 * decimals();
```

Recommendation:

Replace with

maxSupply_ = 2_000_000_000 * 10 ** decimals();

and add a unit-test that asserts maxSupply()==2_000_000_000 * 10 ** 8.

Updates

The Kommunitas team fixed this issue by changing the max supply to the correct value.

SHB.2 mint() is restricted to the admin of MINTER_ROLE, not to the minter itself

- Severity: CRITICAL
- Likelihood: 3
- Status: Fixed
- Impact: 3

Description:

The modifier is: `onlyRole(getRoleAdmin(MINTER_ROLE))` (line 101). With OpenZeppelin's `AccessControl`, the admin of `MINTER_ROLE` defaults to `DEFAULT_ADMIN_ROLE`. Result: the designated minter cannot call `mint()` while the deployer (or any later admin) can.

Files Affected:

SHB.2.1: KommunitasToken.sol

```
98  function mint(address account, uint256 value)
99    public virtual override
100   onlyRole(getRoleAdmin(MINTER_ROLE))
```

Recommendation:

Change the modifier to `onlyRole(MINTER_ROLE)` or document clearly that only the admin may mint.

Updates

The Kommunitas team fixed the issue by updating the modifier to the correct one onlyRole(MINTER_ROLE).

SHB.3 Empty `_authorizeUpgrade()` makes upgrade safety depend on an external modifier

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

Description:

UUPS proxies rely on `_authorizeUpgrade()` to block arbitrary callers. The body is empty and security is delegated solely to the `proxied` modifier from `ProxyAdminManagerUpgradeable`, whose code is not in scope. If that modifier is ever bypassed (e.g., via re-entrancy) an attacker can upgrade the implementation.

Files Affected:

SHB.3.1: KommunitasToken.sol

```
30 function _authorizeUpgrade(address newImplementation)
31   internal virtual override proxied {}
```

Recommendation:

Add an explicit check such as `onlyRole(UPGRADER_ROLE)` or `onlyRole(DEFAULT_ADMIN_ROLE)` and grant that role exclusively to the on-chain proxy-admin.

Updates

The Kommunitas team acknowledged the issue and stated that in the modifier, the sender is validated against proxyAdminAddres

SHB.4 Hard-revert on transfers to the token contract breaks integrations and loses funds

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

Description:

`transfer`, `transferFrom`, and `approve` revert if the destination (or spender) is `address(this)`. Many DeFi protocols legitimately send tokens to the token contract itself (e.g., staking, sushi-bar, burn-and-mint bridges). Accidental transfers are irrecoverable and the token is unusable in such protocols. `increaseAllowance()` is not overridden, so the guard can be bypassed.

Files Affected:

SHB.4.1: KommunitasToken.sol

```
78 if (to == address(this)) {
79   revert KommunitasTokenSelfTransferred(_msgSender(), value);
80 }
```

Recommendation:

Either allow such transfers or provide an owner-gated `sweep()`/`rescue()` function. If the guard is retained, wrap it in a library-wide policy.

Updates

The Kommunitas team resolved this issue by removing all the reverts and adding the rescuable feature.

SHB.5 `pause()` does not stop minting / burning

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

Description:

`transfer()` functions use `whenNotPaused`, but `mint()`, `burn()`, and `burnFrom()` (inherited) do not. During an incident an attacker with minter privileges could still change total supply.

Recommendation:

Add `whenNotPaused` to all supply-changing functions, or include the pause check in `_beforeTokenTransfer()`.

Updates

The Kommunitas team resolved the issue by adding the whenNotPaused modifier in the burn and mint functions.

SHB.6 `setMaxSupply()` can raise the cap to an arbitrary value

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

Description:

There is no upper bound; an admin could inflate supply far beyond what token holders expect.

Files Affected:

SHB.6.1: KommunitasToken.sol

```
116 function setMaxSupply(uint256 newMaxSupply_)
117   public virtual
118   onlyRole(getRoleAdmin(DEFAULT_ADMIN_ROLE))
```

Recommendation:

Either remove the setter or enforce a project-approved hard limit (e.g., `newMaxSupply_ <= 2_000_000_000 * 10**8`).

Updates

The Kommunitas team has resolved this issue by adding a verification in the setMaxSupply function. If the condition is not met, the revert MaxSupplyReached is triggered.

SHB.7 `KommunitasTokenSelfTransferred` uses the spender instead of the token-owner

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

Description:

`transferFrom()` reverts with `KommunitasTokenSelfTransferred(_msgSender(), value)`. The error's first parameter is meant to be the `from` address, not the spender.

Recommendation:

Change to `KommunitasTokenSelfTransferred(from, value)`.

Updates

The Kommunitas team has resolved this issue by removing the revert if destination/spender is address(this).

SHB.8 No storage-gap reserved for future upgrades

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

Description:

Upgradeable contracts should end with `uint256[50] private __gap;` to avoid storage-layout collisions in later versions.

Recommendation:

Append the gap array at the end of the contract.

Updates

The Kommunitas team has mitigated this risk by implementing the upgrade in a new smart contract that inherits from the existing parent contract. This approach ensures safety and prevents storage collisions.

SHB.9 No way to rescue accidentally sent ERC-20 or Ether

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

Description:

Users (or integrations) may mistakenly send assets to the contract; they are locked forever.

Recommendation:

Provide admin-gated `rescueERC20()` and `rescueETH()` functions.

Updates

The Kommunitas team resolved this issue by adding the rescuable feature.

SHB.10 `ERC20Permit` cached domain-separator breaks on chain-ID fork

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

Description:

`ERC20PermitUpgradeable` stores the initial `chainId` forever. If the chain hard-forks, all pre-fork permits become invalid.

Recommendation:

Override `DOMAIN_SEPARATOR()` per OZ 4.9 guidelines to recompute the separator if `block.chainid` changes.

Updates

The Kommunitas team stated that they are already using OZ 5.3.0 and have already constructed the DOMAIN_SEPARATOR on the fly.

SHB.11 Self-transfer guard can be bypassed via `increaseAllowance()`

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

Description:

`approve()` is guarded, but `increaseAllowance()` and `decreaseAllowance()` are inherited unmodified, allowing the user to set a non-zero allowance for `address(this)`.

Recommendation:

Override both functions with the same self-transfer check or remove the guard entirely.

Updates

The Kommunitas team has resolved this issue by removing the revert if destination/spender is address(this).

4 Best Practices

BP.1 `decimals()` can be marked `pure`

Description:

The function `decimals()` returns the constant value 8 and does not read contract state. Marking it `pure` instead of `view` enables a small byte-code and gas refund.

Files Affected:

BP.1.1: KommunitasToken.sol

```
70 function decimals() public pure override returns (uint8) {
71     return 8;
72 }
```

Status - Fixed

The Kommunitas team has resolved the issue by adding the 'pure' keyword.

BP.2 Shorten custom-error names

Description:

Errors like `KommunitasTokenMaxSupplyReached` and `KommunitasTokenSelfTransferred` increase deployment byte-code size. Shorter names (e.g. `MaxSupplyReached`) give identical semantics at lower cost. This is style-level but saves ∼50–100 bytes in the runtime.

Status - Fixed

The Kommunitas team has resolved the issue by shorting the error names.

BP.3 Use `unchecked` arithmetic once bounds are proven

Description:

Inside `mint()` the code already checks that `totalSupply + value ≤ _maxSupply`. The subsequent addition can be wrapped in an `unchecked` block to save ~25 gas.

Files Affected:

BP.3.1: KommunitasToken.sol

```
105 unchecked {
106     _totalSupply += value;
107 }
```

Status - Acknowledged

The team acknowledged the issue since they are using the _mint() function from OZ.

BP.4 Cache `address(this)` in an immutable

Description:

The transfer-to-self guard compares `to == address(this)` on every transfer. Storing the contract's address in an immutable variable `TOKEN_ADDRESS` once saves ~5 gas per call.

Files Affected:

BP.4.1: KommunitasToken.sol

```
20 address immutable TOKEN_ADDRESS = address(this); // set in initializer
21 ...
22 if (to == TOKEN_ADDRESS) {
23     revert SelfTransfer(msg.sender, value);
24 }
```

Status - Fixed

The Kommunitas team has resolved this issue by removing the revert if destination/spender is address(this).

5 Conclusion

In this audit, we examined the design and implementation of Kommunitas Token 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.

6 Scope Files

6.1 Audit

Files	MD5 Hash
token/KommunitasToken.sol	61e55e863c4fdedb05e79217fe30c5cd

6.2 Re-Audit

Files	MD5 Hash
token/KommunitasToken.sol	9948151e94c9a5e49349de9c48c5471c

7 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.

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