# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Date:** 12 September, 2025

# Document

| | |
|---|---|
| **Name** | Smart Contract Code Review and Security Analysis Report for ETHICS (JUSTICE) |
| **Approved By** | Svyatoslav Nadozirny \| Solidity SC Auditor |
| **Auditor company** | Coders Valley Ltd.<br>63-66 Hatton Garden<br>Fifth Floor, Suite 23<br>EC1N 8LE - London<br>London (GB)<br>United Kingdom |
| **Type** | ERC-20 Fungible Token |
| **Platform** | Ethereum Mainnet |
| **Language** | Solidity v0.8.28 (OpenZeppelin 5.x patterns, libs compiled with ^0.8.20) |
| **Methodology** | Referenced document for audit methodology |
| **ChangeLog** | September 12, 2025 – initial release |

# Table of contents

# Introduction

The Customer engaged our company to evaluate the **ETHICS** smart-contract (symbol **JUSTICE**) for security, code quality and compliance with ERC-20 best practices. This report summarizes our findings and provides actionable recommendations.

# Scope

General data provided by the Customer

- Token name: ETHICS
- Token symbol: JUSTICE
- Decimals: 18
- Blockchain: Ethereum
- Contract standard: ERC-20
- Total supply: 100,000,000
- For sale: 50,000,000 (50%)
- Website: https://www.worldethics.info
- Deployment style: **Minimal Proxy** to implementation **UltimateTokenOwnable**

The scope of the project includes the following smart contracts from the link:

**Contracts**: https://etherscan.io/token/0x69dB04251ed748705D50796E9758AB5Dd2b000E9#code

- `UltimateTokenOwnable.sol` - implementation contract combining OpenZeppelin-style ERC-20 with extensions (Ownable, Pausable, Capped, TokenURI) and **Initializable** pattern for minimal proxies.
- OpenZeppelin-based modules (upgradeable patterns using ERC-7201 storage namespaces):
- `Initializable.sol`, `Ownable.sol`, `Pausable.sol`, `ERC20.sol` (modified), `ERC20Capped.sol`, `ERC20TokenMetadata.sol`, plus interfaces (`IERC20.sol`, `IERC20Metadata.sol`, `draft-IERC6093.sol`) and `Context.sol`.

**Deployment model**

- Minimal Proxy (CreateMyToken factory): *"UltimateTokenOwnable"* with `initialize` function (initializer guard).
- Compiler: v0.8.28, Optimizer: **enabled (200 runs)**, EVM: **paris**.

**Live Code**: Provided (source bundle)

**Technical Documentation:** Whitepaper (ETHICS/JUSTICE, 2025) - https://www.worldethics.info/_files/ugd/3d727a_b01fbcfe0a0a4f668ec33ffa80cd8710.docx?dn=WHITEPAPER%204%20ETHICS%20(JUSTICE)%20DEUTSCH.docx - provided

**Tests**: Not provided

**Environment:** Foundry/Forge settings JSON provided (optimizer, remappings, viaIR, EVM Paris)

# SHA256 Hash

SHA256 hash of the source code - not computed in scope of this report.

# Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors. |
| High | High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors. |
| Medium | Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category. |
| Low | Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality. |

# Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

## Documentation quality

The total Documentation Quality Score is 8 out of 10.

- **Functional requirements** are provided in https://www.worldethics.info/_files/ugd/3d727a_b01fbcfe0a0a4f668ec33ffa80cd8710.docx?dn=WHITEPAPER%204%20ETHICS%20(JUSTICE)%20DEUTSCH.docx
  Whitepaper provides vision, tokenomics (50/20/15/10/5), roadmap, and operating principles; public parameters (name, symbol, decimals, total supply, ICO share) are clear. (Score: 5/5).

- **Technical Requirements**: Compiler/EVM/optimizer details available; deployment procedures, test plans, and operational runbooks absent. Foundry settings present, however, precise deployment/initialization parameters (owner, mint target, cap) and addresses for vesting/allocations are not formally documented, no migration/rollback plan. (Score: 3/5).

- **NatSpec Adherence**: Not used — reduces in-code clarity.

## Code quality

The total Code Quality Score is 8 out of 10.

- **Development Environment:**
  - Code relies on OpenZeppelin 5.x patterns with ERC-7201 namespaced storage to mitigate collision risks.
  - Correct `initializer` guard; however, parent `__Ownable_init`/`__Pausable_init` are not invoked (see Low-severity note). (Score: 3/5).
- **Solidity Style Guide Compliance**: Consistent formatting; clear separation of concerns (ownership, pausing, capping, metadata). (Score: 5/5).
- Areas to improve: prefer `__Ownable_init(_owner)` and `__Pausable_init()` instead of direct `_transferOwnership`; add `require` checks for zero addresses in `initialize`; add event for tokenURI change, extend tests and CI.

## Security score

The security Score is 9 out of 10.

All previously identified critical and high-severity issues have been remediated. However, several low-severity issues persist, and there is still no automated test coverage (0 % branch coverage), which prevents a perfect security rating. (Score: 9/10).

- **Critical Issues**: None
- **High Issues**: None
- **Medium Issues**: 1
- **Low Issues**: 3
- **Notes:** Centralized minting under `onlyOwner` up to the cap, ensure governance/ops controls.

## Summary

According to the assessment, the Customer's smart contract has the following score: **8.7**.

The system users should acknowledge all the risks summed up in the risks section of the report.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Final score | |

**Breakdown:**

- Documentation Quality: 8/10

- Code Quality: 8/10
- Security Level: 9/10
- Test Coverage: Not provided (requires unit tests for scoring).

*Note: The final score is weighted according to the methodology (Documentation weighted at 1.0, Code Quality at 2.0, Security at 7.0), and the absence of unit tests impacts the overall score.*

**Table. The distribution of issues during the audit**

| Review date | Low | Medium | High | Critical |
|---|---|---|---|---|
| 12 September, 2025 | 3 | 1 | 0 | 0 |

# Risks

No exploitable vulnerabilities were identified in core token mechanics. The implementation closely follows OpenZeppelin ERC-20 patterns with a supply **cap** and **pause** guard applied to the unified `_update` path (covers transfer/mint/burn).

Residual risks are operational and governance-related:

- **Centralization risk:** `onlyOwner` can pause the token and mint up to the configured cap. If the deployer retains EOA ownership, compromise of the owner key can impact token operations.

- **Misconfiguration risk:** The supply cap (`_maxSupply`) is provided at initialization. If configured above the communicated total supply (100,000,000), additional issuance up to the cap becomes possible.

- **Operational risk while paused:** With `_update` gated by `whenNotPaused`, pausing also blocks *mint* and *burn*. Ensure documented procedures for incident response and resumption.

# System Overview

**Contract:** `UltimateTokenOwnable` (Minimal Proxy instance)
**Standard:** ERC-20
**Extensions:** Ownable, Pausable, Capped Supply, Token Metadata.

**Key behaviors**

- **Initialization (`initialize`)**
  - Sets name/symbol/decimals via `__ERC20_init`.
  - Initializes supply cap via `__ERC20Capped_init(_maxSupply)`.

    - Transfers ownership with `_transferOwnership(_owner)`.
    - Mints `_initialSupply` to `_mintTarget`.
    - Sets token metadata URI with `_setTokenUri(tokenUri_)`.
- **Transfer/Mint/Burn path**
    - All state transitions route through `_update` (OpenZeppelin 5.x).
    - Contract overrides `_update` with `whenNotPaused`, pausing **halts transfers, mints, and burns**.
    - `ERC20Capped` post-mint check reverts if `totalSupply()` would exceed `cap()`.
- **Administrative controls**
    - `pause()` / `unpause()` — only owner.
    - `mint(address to, uint256 amount)` — only owner; subject to cap.
    - `burn(uint256 value)` — self-burn by holder.
    - `setTokenURI(string tokenUri_)` — only owner.
- **Events & errors**
    - Standard `Transfer/Approval`, `Paused/Unpaused`, `OwnershipTransferred`, and OZ ERC-6093 custom errors (e.g., `ERC20InsufficientBalance`).

# Privileged roles

- **Owner** (EOA or multisig recommended):
    - `pause()` / `unpause()` — globally disables/enables all token movements.
    - `mint(address to, uint256 amount)` — mints subject to `cap`.
    - `setTokenURI(string)` — updates off-chain metadata pointer.
    - `transferOwnership(address)` / `renounceOwnership()`.

No other elevated roles exist (no separate `MinterRole`, no blacklist/fees).

# Recommendations

To further enhance the quality and maintainability of the ETHICS contract, the following recommendations are made:

1. **Supply Policy Enforcement (Medium)**

   Ensure the configured cap (`_maxSupply`) equals the communicated maximum supply (**100,000,000 · 10^18**). Lock this policy in project docs and distribution schedules. Consider emitting an event in `initialize` summarizing `cap`, `initialSupply`, and `mintTarget` for off-chain indexers.

2. **Ownership Hardening (Low)**

- Validate `_owner != address(0)` inside `initialize` to avoid accidental renounce at deployment.
- Transfer ownership to a **multisig** (e.g., Gnosis Safe) and consider a **Timelock** for mint/pause operations.
- Publish an **owner actions policy** (what actions are allowed and when) to reduce centralization concerns.

3. **Initialization Hygiene (Low)**
   Although functional correctness is unaffected, call parent initializers to stay aligned with OZ upgradeable conventions and future-proof upgrades:

```
function initialize(...) external initializer {
__ERC20_init(_name, _symbol, _decimals);
__ERC20Capped_init(_maxSupply);
__Pausable_init();
__Ownable_init(_owner);
_mint(_mintTarget, _initialSupply);
__ERC20TokenMetadata_init(tokenUri_); // or keep _setTokenUri
}
```

   This also enforces the non-zero owner check from `__Ownable_init`.

4. **Operational Runbooks & Monitoring (Low)**

- Document procedures for emergency **pause** and subsequent **unpause**.
- Publish distribution plan for the **50% sale allocation** and any **vesting/lockups**.
- Add on-chain monitors/alerts for `OwnershipTransferred`, `Paused`, `Unpaused`, and `mint` calls.

5. **Testing & CI (Informational)**

- Provide unit tests (e.g., Foundry/Hardhat) for mint capping, pausing semantics (transfers/mints/burns blocked), ownership transitions, and metadata updates.
- Add static analysis (Slither) and coverage reporting in CI.

## Checked Items

The contract was audited for commonly known and specific vulnerabilities. Here is a summary of the items considered:

| Item | Type | Description | Status |
|---|---|---|---|
| **Default Visibility** | SWC-100 SWC-108 | Functions and state variables visibility should be set explicitly. | Passed |
| **Integer Overflow and Underflow** | SWC-101 | Solidity ^0.8.x includes built-in overflow and underflow protection. | Passed |
| **Outdated Compiler Version** | SWC-102 | Uses recent Solidity version ^0.8.30. | Passed |
| **Floating Pragma** | SWC-103 | Contracts should deploy with a fixed compiler version. | Passed |
| **Unchecked Call** | SWC-104 | Ensures the return value of calls is checked. | Not Relevant |

| | | | |
|---|---|---|---|
| **Return Value** | | | |
| **Access Control & Authorization** | CWE-284 | Properly implemented without unauthorized access to protected functions. | Passed |
| **SELFDESTRUCT Instruction** | SWC-106 | Contract does not contain self-destruct functionality. | Not Relevant |
| **Check-Effect-Interaction** | SWC-107 | Follows the pattern to prevent reentrancy attacks.. | Passed |
| **Assert Violation** | SWC-110 | Proper code execution prevents reaching a failing assert statement. | Passed |
| **Deprecated Solidity Functions** | SWC-111 | No deprecated functions are used. | Passed |
| **Delegatecall to Untrusted Callee** | SWC-112 | No delegatecall usage to untrusted addresses. | Not Relevant |
| **DoS (Denial of Service)** | SWC-113 SWC-128 | No risks of DoS attacks through contract design. | Passed |
| **Race Conditions** | SWC-114 | No race conditions or transaction order dependencies identified. | Informational |
| **Authorization through tx.origin** | SWC-115 | tx.origin should not be used for authorization. | Passed |
| **Block values as a proxy for time** | SWC-116 | Block numbers are not used as time proxies. | Passed |
| **Signature Unique Id** | SWC-117 SWC-121 SWC-122 EIP-155 | Not applicable, as the contract does not use message signatures.. | Not Relevant |
| **Shadowing State Variable** | SWC-119 | State variables are not shadowed. | Passed |
| **Weak Sources of Randomness** | SWC-120 | Randomness is not generated using block attributes. | Not Relevant |
| **Incorrect Inheritance Order** | SWC-125 | Inheritance order is carefully specified. | Passed |
| **Calls Only to Trusted Addresses** | EEA-Level-2 SWC-126 | External calls are only performed to trusted addresses. | Passed |

| | | | |
|---|---|---|---|
| **Presence of unused variables** | [**SWC-131**]() | The code should not contain unused variables if this is not [justified]() by design. No unused variables found, ensuring efficient code. | Passed |
| **EIP standards violation** | [**EIP**]() | The contract adheres to EIP standards, particularly ERC-20. | Passed |
| **Assets integrity** | **Custom** | Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract. | Passed |
| **User Balances manipulation** | **Custom** | Contract owners or any other third party should not be able to access funds belonging to users. | Passed |
| **Data Consistency** | **Custom** | Smart contract data should be consistent all over the data flow. | Passed |
| **Flashloan Attack** | **Custom** | When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. | Not Relevant |
| **Token Supply manipulation** | **Custom** | Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer. | Warning |
| **Gas Limit and Loops** | **Custom** | Code is optimized to avoid high gas usage and unbounded loops. | Passed |
| **Style guide violation** | **Custom** | Style guides and best practices should be followed. | Passed |
| **Requirements Compliance** | **Custom** | The code should be compliant with the requirements provided by the Customer. | Passed |
| **Environment Consistency** | **Custom** | The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | Not Relevant |
| **Secure Oracles Usage** | **Custom** | The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles. | Not Relevant |
| **Tests Coverage** | **Custom** | The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by | **Failed** |

| | | | |
|---|---|---|---|
| | | multiple users should be tested. | |
| **Stable Imports** | **Custom** | The code should not reference draft contracts, that may be changed in the future. | Passed |

# Findings

## Critical

No issues

## High

No issues

## Medium

**M-1 — Supply policy may diverge from communicated total supply if cap misconfigured**
**Description:** The cap is provided at initialization. If `_maxSupply` is set above the communicated max supply (100,000,000), the owner can mint additional tokens up to the cap, diverging from public expectations.

**Recommendation:** Set `cap == 100,000,000 · 10^18`, document it publicly, and consider emitting an event in `initialize` with `cap` and `initialSupply`. Optionally hard-code the cap if policy is immutable.

## Low

**L-1 — Missing zero-address validation for `_owner` in `initialize`**
**Description:** `initialize` uses `_transferOwnership(_owner)` without checking for `address(0)`. Passing zero would renounce ownership at deployment and disable `onlyOwner` functions. While not a vulnerability per se, it is a foot-gun.

**Recommendation:** Require `_owner != address(0)` or call `__Ownable_init(_owner)` which enforces this.

**L-2 — Parent initializers not called**
**Description:** `__Pausable_init` and `__Ownable_init` are not invoked. Defaults are correct today (`paused == false`), but adhering to OZ initializer patterns better future-proofs upgrades and explicit state.

**Recommendation:** Call parent initializers in `initialize` (see §9.3 snippet).

**L-3 — Centralized administration**
**Description:** The design intentionally centralizes minting and pausing to the owner. While acceptable for many tokens, it concentrates power and creates a single point of failure.

**Recommendation:** Migrate ownership to a multisig; consider a timelock for sensitive ops; publish governance policy.

# Disclaimers

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications.

Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

## Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.