**# Mock Audit #1**

---

**### Scope Notes**

**\*\*Repo:\*\*** OpenZeppelin Contracts v4.x

**\*\*Commit Hash:\*\*** 4a67c6f3ab2be41487889a020c99e11dedbd6eb4

**\*\*Target:\*\*** ERC20 token implementation

**\*\*Contracts in Scope:\*\***

- ERC20.sol (main implementation, 305 LoC)

- IERC20.sol (interface – referenced only)

- Context.sol (utility – no audit focus)

- IERC20Metadata.sol (metadata interface – low risk)

**\*\*Out of Scope:\*\***

- Extensions (ERC20Burnable, ERC20Capped, ERC20Pausable, etc.)

- Upgradeable variants (ERC20Upgradeable.sol)

- Any other unrelated utilities/helpers

---

**### Severity Criteria**

**\*\*High\*\***

- Direct loss of funds or permanent lock of assets.

- Anyone can exploit (not just privileged roles).

- Breaks core protocol functionality.

**\*\*Medium\*\***

- Causes significant disruption (DoS, griefing, governance failure).

- Exploitable under some conditions or requires privileged roles.

- Financial loss is possible but limited.

**\*\*Low\*\***

- Minor issues: inefficiencies, gas waste, unclear logic, small inconsistencies.

- Doesn't threaten core security or funds.

**\*\*Informational / Non-Critical\*\***

- Code style, readability, missing comments.

- Best practices (naming conventions, event emissions, input validation improvements).

- No security impact.

---

**## Findings**

**### High Severity**

**### Medium Severity**

**\*\*M-01: ERC20 Approval Race\*\***

*\*Summary\**

Changing a non-zero allowance with `approve(spender, new)` can be raced by the spender using `transferFrom()`, allowing use of both the old and new values under tx ordering.

*\*Location\**

`ERC20.approve`, `ERC20.transferFrom`, `ERC20.\_approve`, `ERC20.\_spendAllowance`

*\*Description\**

`approve` overwrites `\_allowances[owner][spender]` without requiring `value` to be zero first.  `transferFrom()` spends whatever allowance is current at the execution (No atomic link between "change" and "spend").

*\*Proof of Concept (PoC)\**

1.  Deploy an ERC20 token.

2.  `owner` approves `spender` for 100 tokens:

    `token.approve(spender, 100);`

3.  Later, `owner` wants to reduce to 20, so broadcasts:

    `token.approve(spender, 20);`

4.  Before that tx is mined, `spender` front-runs with:

    `token.transferFrom(owner, attacker, 100);`

    (uses old allowance).

5.  Then the `approve(20)` tx lands, setting a new allowance.

6.  `spender` can now call:

    `token.transferFrom(owner, attacker, 20);`

    (uses new allowance).

Result: Spender drains 120 tokens instead of the intended 20.

*\*Impact\**

Owner's intended reduction/reset can be bypassed during the race window, therefore effective spend may be "old + new".

*\*Severity\**

Medium: known ERC-20 limitation; the exploit requires a motivated spender + mempool timing but results in unexpected extra spend.

*\*Recommendations\**

- Two-step change: `approve(spender, 0)` -> wait confirmation -> `approve(spender, new)`.

- Consider EIP-2612/Permit2 flows to avoid multi-tx races.

*\*Status\**

Known ERC-20 standard limitation, not an implementation bug.

**### Low Severity**

**### Informational / Non-Critical**

**\*\*L-01: Inconsistent pragma directives\*\***

*\*Description\**

FIles within the ERC20 implementation use different pragma constraints (`>=0.4.16`, `>=0.6.2`, `>=0.8.4`, `>=^0.8.20`).

*\*Impact\**

Low.  No direct vulnerability, but compilation differences could introduce small discrepencies across environments.

*\*Recommendation\** Standardise pragma directives (e.g., `^0.8.20`) across all contracts).

---

**## Access Control Review**

**\*\*Observation\*\***

All external functions are appropriately unrestricted for an ERC20 token Implementation.

This is expected behaviour - ERC20 is a public token standard, and core functions like `transfer()`, `approve()`, and `transferFrom()` are designed for public use.

**\*\*Justification\*\***

- All functions operate on `msg.sender`'s funds or rely on allowances explicitly set by token owners.

- No sensitive administrative actions (e.g., minting, burning, pausing) are present in this contract.

- Therefore, no `onlyOwner`, role-based access control, or modifiers are required.

Conclusion: No access control vulnerabilities found.

---

**## State Management**

- All state updates are routed through the internal `\_update` and `\_approve` functions.

- The contract uses Solidity 0.8+ built-in overflow checks for minting (`\_totalSupply += value`).  All other arithmetic is wrapped in `unchecked` blocks, but is safe due to prior guards.

- Every state change correctly emits the matching `Transfer` or `Approval` event.

Conclusion: State handling is consistent with ERC-20 standard, and includes no vulnerabilities.

---

**## Function Review**

**\*\*External\*\***

`transfer(address to, uint256 value)` - Debits sender, credits recipient via `\_update`, emits `Transfer`. No reentrancy risk.

`approve(address spender, uint256 value)` - Sets allowance, subject to approval race (Finding M-01).

`transferFrom(address from, address to, uint256 value)` - Spends allowance via `\_spendAllowance`, same race caveat.

`name()` - Returns the token name from storage. Read-only; no external calls; no state change; no reentrancy risk.

`symbol()` - Returns the token symbol from storage. Read-only; no external calls; no state change; no reentrancy risk.

`decimals()` - Returns the decimals metadata. Read‑only; no external calls; no state change; no reentrancy risk.

`totalSupply()` - Returns `totalSupply`. Read‑only; no external calls; no reentrancy risk.

`balanceOf(address account)` - Returns `\_balances[account]`. Read‑only; no external calls; no state change; no reentrancy risk.

`allowance(address owner, address spender)` — Returns the allowance from `\_allowances[owner][spender]`. Read-only; no external calls; no state change; no reentrancy risk.

**\*\*Internal\*\***

`\_update(from, to, value)` — Single entry point for transfer/mint/burn; enforces balances; updates `\_totalSupply`; emits `Transfer`.

`\_spendAllowance(owner, spender, value)` — Reverts if insufficient; skips decrement when allowance is `type(uint256).max`.

`\_approve(owner, spender, value, bool emitEvent)` — Writes allowance; optionally emits `Approval`.

---

**## Static Analysis Findings**

Tool: Slither Analyzer

Target: contracts/token/ERC20/ERC20.sol

Compiler: solc 0.8.20

Result:

- 1 issue detected across 7 contracts with 46 detectors.

- Finding: Different pragma directives are used.

A screenshot of a computer program

AI-generated content may be incorrect.