# RISK AUDIT

for

# CraftEngine

on

May 31, 2025

FIDESIUM

# Executive Summary

## Report

43
/100

TOTAL

Medium risk

May 31, 2025

## Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on CraftEngine **contract**, provided as a flat SOL file

Contract File: TaxToken.sol

## Issue Summary

| Critical | High | Medium | Low | Info |
|---|---|---|---|---|
| 2 Issues | 2 Issues | 6 Issues | 1 Issues | 2 Issues |

## Caveats

CraftTokens's codebase is generally well written, but does incur a handful of high risk flaws.

## Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

## Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

## Severity Definitions

| | |
|---|---|
| Critical | The issue can cause large economic losses, large-scale data disorder or loss of control of authority management. |
| High | The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications. |
| Medium | The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact. |
| Low | The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business. |
| Informational | The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth. |

## Risk Issues

| Vulnerability | Description | Risk | Probability | Status |
|---|---|---|---|---|
| Centralization: No renouncement | Owner can not renounce. | Critical | Critical | Active |
| DoS: Large Contract Balances | Balance accumulation can lead to protocol failure | Critical | Critical | Active |
| Incorrect transfer call | The `swapBack` function calls `transfer` | High | High | Active |
| Reentrancy | The `swapBack` function allows reentrancy | High | Low | Active |
| Centralization | Owner has significant modification rights over the contracts and their state. | Medium | Medium | Active |
| Unlimited Token Approval | Unlimited tokens are approved. | Medium | Medium | Active |
| Sandwich Attack: No Slippage protection | The contract swaps tokens without slippage protection | Medium | Medium | Active |
| Economic Attack: Fee bypass | `setAutomatedMarketMakerPair` can lead to fee bypass | Medium | Low | Active |
| Reliance on Block Timestamp | Multiple functions rely on `block.timestamp`, which can be manipulated by miners. | Medium | Low | Active |
| One Step Ownership Transfer | The contract applies the `Ownable` pattern. It relies on a one step `transferOwnership` strategy. This exposes the contracts to accidental ownership transfer to malicious or invalid wallets | Medium | Low | Active |
| Missing constant assignment | Multiple variables have no setters and should be constants | Low | Low | Active |
| Unused import | `ERC20Burnable` is imported but never used | Info | Info | Active |
| Gas Optimization: Inefficient Storage Access | Several locations have storage access inefficiencies | Info | Info | Active |

## Vulnerabilities Critical

### Centralization: No renouncement

Vulnerability severity: **Critical**

Vulnerability probability: **Critical**

Owner can not renounce.

Given this is a tax driven ERC20 with marketing wallet, communities frequently expect owner renouncement to avoid catastrophic rugpulls

Recommendations:

Add at least a partial renouncement of dangerous functions

```
mapping(bytes4 => bool) public renouncedFunctions;
bool public criticalFunctionsLocked = false;
event FunctionRenounced(bytes4 indexed functionSelector);
event CriticalFunctionsLocked();
...
function lockCriticalFunctions() external onlyOwner {
    require(launched, "Cannot lock before trading is enabled");
    require(!criticalFunctionsLocked, "Already locked");

    // Lock dangerous functions
    renouncedFunctions[this.setExcludedFromFees.selector] = true;
    renouncedFunctions[this.withdrawStuckToken.selector] = true;
    renouncedFunctions[this.withdrawStuckETH.selector] = true;
    renouncedFunctions[this.setFeesEnabled.selector] = true;

    criticalFunctionsLocked = true;
    emit CriticalFunctionsLocked();
}
...
function renounceFunction(bytes4 functionSelector) external onlyOwner {
    require(!renouncedFunctions[functionSelector], "Already renounced");
    renouncedFunctions[functionSelector] = true;
    emit FunctionRenounced(functionSelector);
}
...
modifier notRenounced() {
    require(!renouncedFunctions[msg.sig], "Function has been renounced");
    _;
}
...
function withdrawStuckToken(address token, address to) external onlyOwner notRenounced {
    ...
```

## Vulnerabilities Critical

### DoS: Large Contract Balances

Vulnerability severity: **Critical**

Vulnerability probability: **Critical**

Balance accumulation can lead to protocol failure

swapBack processes `swapTokensAtAmount * swapCapMultiplier`

Given the reliance on both volume and owner set environment variables, large balance accumulations could lead to critical delays across an unknown number of blocks

Recommendations:

Implement percentage based calculations based on balance

## Vulnerabilities High

### Incorrect transfer call

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `swapBack` function calls `transfer`

```
transfer(marketingWallet, tokensForMarketing);
transfer(teamWallet, tokensForTeam);
```

Calls to `transfer` will incur fees, and potentially cause recursive swaps

Recommendations:

Use `_transfer`

### Reentrancy

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `swapBack` function makes external calls without reentrancy protection.

```
transfer(marketingWallet, tokensForMarketing);
transfer(teamWallet, tokensForTeam);
```

While direct reentrancy is protected from via the `swapping` variable. More sophisticated contract manipulation and/or cross functional reentrancy is possible due to the direct `transfer` calls

Recommendations:

Add a reentrancy guard

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
...
contract TaxToken is ERC20, Ownable, ReentrancyGuard {
...
function _transfer(address from, address to, uint256 amount)
internal
override
nonReentrant
{
```

## Vulnerabilities Medium

### Centralization

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

Owner has significant modification rights over the contracts and their state.

These include, but are not limited to totally draining the contract via `withdrawStuckETH` and manipulating the fee structure via `setExcludedFromFees`

Recommendations:

Implement timelock, and/or multisig governance. At the very least ensure that the high value Owner wallet is a well managed multisig.

### Sandwich Attack: No Slippage protection

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The contract swaps tokens without slippage protection

This can lead to Sandwich Attacks and/or MEV bot abuse.

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
tokensToSwap, 0, path, address(this), block.timestamp  // minAmountOut = 0
);
```

Recommendations:

Implement reasonable slippage protection. Fidesium typically recommends the 5-10% range.

### Unlimited Token Approval

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

Unlimited tokens are approved.

If the router is compromised this could lead to total token drain and therefore total protocol failure

```
_approve(address(this), address(uniswapV2Router), type(uint256).max);
```

Recommendations:

Approve only necessary amounts before each swap, and remove global approval

## Vulnerabilities Medium

### One Step Ownership Transfer

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

The contract applies the `Ownable` pattern. It relies on a one step `transferOwnership` strategy. This exposes these contracts to accidental ownership transfer to malicious or invalid wallets.

Recommendations:

Implement `Ownable2Step` to drive a two step ownership transfer.

### Economic Attack: Fee bypass

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`setAutomatedMarketMakerPair` can lead to fee bypass

Fake or malicious address could be set which wouldn't trigger fee collection

Recommendations:

Restrict AMM pair addition or implement whitelist of approved DEX factories.

### Reliance on Block Timestamp

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Multiple functions rely on `block.timestamp`, which can be manipulated by miners.

Recommendations:

Rely on a combination of `block.timestamp` and `block.number`, or an external time Oracle.

## Vulnerabilities Low

### Missing constant assignment

Vulnerability severity: **Low**

Vulnerability probability: **Low**

Multiple variables have no setters and should be constants

- marketingFee
- teamFee
- liquidityAmount

Recommendations:

Ensure these variables either have setters with appropriate validations, or are constants

# Vulnerabilities Info

## Unused import

Vulnerability severity: **Info**

Vulnerability probability: **Info**

`ERC20Burnable` is imported but never used

Recommendations:

Remove unused imports

## Gas Optimization: Inefficient Storage Access

Vulnerability severity: **Info**

Vulnerability probability: **Info**

Several locations have storage access inefficiencies

Recommendations:

- Cache storage lookup in `swapBack`

```
uint256 _swapTokensAtAmount = swapTokensAtAmount;
uint256 _swapCapMultiplier = swapCapMultiplier;
uint256 maxSwap = _swapTokensAtAmount * _swapCapMultiplier;
```

- Cache Mapping Lookups in `_transfer`

```
bool isToAMM = automatedMarketMakerPairs[to];
bool isSell = isToAMM && !swapping;
...
if (isToAMM && sellTotalFees > 0) {
```

- Use immutable for constructor set values

```
IUniswapV2Router02 public immutable uniswapV2Router;
```

# Disclaimer

## Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.