ACST: Smart Contract Audit & Code Review

Prepared for: ACST & Dr. Crypto
Date: May 17, 2024

Table of Contents

Disclaimer

This document might include private information regarding the ACST Token (the "Company") intellectual property and IT systems, as well as details on potential weaknesses and ways to exploit them.

After all vulnerabilities have been patched, the Company may decide whether to use the secret information in the report internally or to make it public.

Audit Information

Name	ACST
Type of Contracts	ERC20 Tax Token
Platform	Polygon
Language	Solidity
Testing Methods	AI Analysis, Bot Scan, Static Analysis, and Manual Review
Website	https://polygonscan.com/address/0x533336f16ccc30423eb25822b546a11172263864#code
Date	May 17, 2024

Executive Summary

AuditBase has completed an audit for ACST, a taxable ERC20 token on Polygon. The scope of this audit was limited to the token and tax functionality.

Scope

Deployed Contract	https://polygonscan.com/address/0x533336f16ccc30423eb25822b546a11172263864#code
Documentation	N/A
Unit Tests	N/A
Contracts	polygon_acst_sell_tax.sol

Security Definitions

Risk	Description
Critical	Critical flaws can result in the loss of assets or the alteration of data and are typically simple to exploit.
High	High-level vulnerabilities are challenging to attack, but they can have a big impact on how smart contracts work, like giving the public access to essential features.
Medium	Although medium-level vulnerabilities should be fixed, they cannot result in the loss of assets or the manipulation of data.
Low	Low-level flaws are typically caused by bits of unneeded, old code that don't have a big influence on the execution.

Findings

Inaccurate Balance Update on TransferFrom

The transferFrom function inaccurately updates the balances[taxReceiver] without adding to the existing balance, potentially overwriting it.

Severity: Critical

```
Java
balances[taxReceiver] = fee;
```

Missing Tax Application in Transfer Function

The tax logic applied in transferFrom is not present in the transfer function, leading to inconsistent tax enforcement.

Severity: Critical

```
Java
// No tax logic in transfer function
```

Insufficient Balance Check in Transfer Function

The transfer function requires that the sender's balance be strictly greater than the amount they wish to send. This logic prevents sending an amount equal to the sender's balance.

Severity: High

```
Java
require(senderBalance > amount, "ERC20: insufficient balance");
```

Potential Reentrancy in TransferFrom

The contract does not employ the Checks-Effects-Interactions pattern, potentially making it vulnerable to reentrancy attacks, especially since it interacts with an external address (taxReceiver) without ensuring reentrancy guard.

Severity: High

```
JavaScript
balances[from] -= amount;
balances[to] += amount;
allowances[from][msg.sender] -= amount;
emit Transfer(from, to, amount);
```

Potential for Integer Overflow

Although Solidity 0.8.x introduces built-in overflow checks, the explicit checks for overflow conditions in mathematical operations or in-depth validation for inputs that could lead to overflows are lacking, specifically in tax calculation.

Severity: Medium

```
Java
uint fee = (amount * taxCollected) / 100;
```

Lack of Event Emission after Tax Collection

The contract does not emit a specific event after collecting taxes in the transferFrom function, which might affect transparency and tracking of tax transactions.

Severity: Medium

```
Java
balances[taxReceiver] += fee;
```

Owner Privileges and Potential Centralization

The setNewTax function allows the contract owner to change the tax rate arbitrarily, which could be misused if not governed properly.

Severity: Medium

```
Java
function setNewTax(uint8 newTax) external onlyOwner returns (uint256)
```

Code Scan Findings

M001 - The owner is a single point of failure and a centralization risk:

Having a single EOA as the only owner of contracts is a large centralization risk and a single point of failure. A single private key may be taken in a hack, or the sole holder of the key may become unable to retrieve the key when necessary. Consider changing to a multi-signature setup, or having a role-based authorization model.

Severity: Medium

Java
File: ACST_token.sol

```
25      function setNewTax(uint8 newTax) external onlyOwner returns (uint256) {
```

https://polygonscan.com/address/0x533336f16ccc30423eb25822b546a11172263864#code/tree/main/#L25:25

L001 - Use Ownable2Step rather than Ownable:

Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

Severity: Low

Java
File: ACST_token.sol

```
6     contract ACST is IERC20, Ownable(msg.sender) {
```

https://polygonscan.com/address/0x533336f16ccc30423eb25822b546a11172263864#code/tree/main/#L6:6

L002 - Setters should have initial value check:

Setters should have initial value check to prevent assigning wrong value to the variable. Assginment of wrong value can lead to unexpected behavior of the contract.

Severity: Low

Java

File: ACST_token.sol

```
25      function setNewTax(uint8 newTax) external onlyOwner returns (uint256) {
26          require(newTax < 100, "Tax percentage cannot exceed 100");
27          taxCollected = newTax;
28          return taxCollected;
29      }
```

https://polygonscan.com/address/0x533336f16ccc30423eb25822b546a11172263864#code/tree/main/#L25:29

L003 - Governance functions should be controlled by time locks:

Governance functions (such as upgrading contracts, setting critical parameters) should be controlled using time locks to introduce a delay between a proposal and its execution. This gives users time to exit before a potentially dangerous or malicious operation is applied.

Severity: Low

Java
File: ACST_token.sol

```
25      function setNewTax(uint8 newTax) external onlyOwner returns (uint256) {
```

Disclaimers

AuditBase Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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).

Regarding the code's security, the audit offers no claims or guarantees. It also cannot be taken into account as an adequate evaluation of the code's usefulness and safety, its bug-free status, or any other contract clauses. Despite the fact that we did our best in conducting the analysis and putting together this report, it is crucial to note that you shouldn't rely solely on it. Instead, we advise moving forward with a number of independent audits and a public bug bounty program to ensure the security of smart contracts.

This audit does not constitute a formal partnership or relationship with the Company and should not be used to assume a business relationship.

Technical Disclaimer

On a blockchain network, smart contracts are set up and carried out. Hacking vulnerabilities may exist in the platform, the programming language used, and other applications used in conjunction with the smart contract. The explicit security of the audited smart contracts cannot therefore be guaranteed.