Report.md

Smart Contract Audit Report

Introduction

A smart contract security review of the **Upgradeable AxirToken contract** was done by **Piyush shukla**, with a focus on the security aspects of the application's smart contracts implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Auditor

Piyush shukla,is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Curretly he's Secured to 3 Hacker Rank globally in Smart Contract Security Platform) or reach out on Twitter Piyushshukla__

About ProtocolName

explanation what the protocol does, some architectural comments, technical documentation

Observations

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

review commit hash **- Upgradeable AxirToken**

fixes review commit hash **-**

Scope

The following smart contracts were in scope of the audit:

- `UpgradeableAxirToken`

Findings Summary

ID	Title	Severity	Status
[H-01]	Centralization Risk : If owner Account was compromised .then all function are useless	High	Acknowledge
[M-01]	whenNotPaused modifier always be first modifier	Medium	Fixed

ID	Title	Severity	Status
[M-02]	Missing zero check in recipient address	Medium	Fixed
[L-01]	Unused library should be remove	low	Fixed
[I-01]	Recommended to use Safemath library	informative	Acknowledge

Detailed Findings

[H-01]Centralization Risk : If owner Account is compromised then it make huge disaster to contract

Smart contract lies in the fact that only the contract owner (the entity that deployed the contract) has the authority to mint and burn tokens. This means that the owner has full control over the token supply and can manipulate it at will. Which make more centralized this contract, if anypoint owner loose the control , or malicious owner takeover the control of owner. it is big disaster to contract

Mitigation

To mitigate this centralization issue, the contract could implement a multi-role approach. Instead of having only the contract owner perform minting and burning, the contract could define multiple roles with specific permissions. For example, separate roles for minting and burning could be created, and these roles could be assigned to different addresses. This way, the power to create and destroy tokens is distributed among different entities, reducing the risk of abuse by a single party.

Add these modifier in code

```
modifier onlyMinter() {
    require(msg.sender == minter, "Not a minter");
    _;
}

modifier onlyBurner() {
    require(msg.sender == burner, "Not a burner");
```

```
        _;
    }
```

And update `mint and burn` function like this

```
  function mint(address _userAddress, uint256 _amount) public whenNotPaused `onlyMinte
      require(_userAddress != address(0), "Invalid user address");
      require(totalSupply() + _amount <= maxTokenSupply, "Exceeds max supply");
      _mint(_userAddress, _amount);
  }

  function burn(uint256 _amount) public whenNotPaused `onlyBurner` {
      _burn(msg.sender, _amount);
  }
```

By implementing these changes, you distribute the authority to mint and burn tokens among different addresses with specific roles, reducing the centralization risk and increasing the overall decentralization of the token system.

Severity

High

Status

Acknowledge

[M-01] whenNotPaused modifier always be first modifier

Always whenNotPaused modifier to implement before any other modifier

correct the code in every function

Use this

```
  function setTreasury(address _treasury) external whenNotPaused onlyOwner  {
         require(_treasury != address(0), "Invalid treasury address");
         treasuryContract = _treasury;
```

```
        emit TreasuryChanged(_treasury);
    }
```

instead

```
  function setTreasury(address _treasury) external onlyOwner whenNotPaused  {
        require(_treasury != address(0), "Invalid treasury address");
        treasuryContract = _treasury;
        emit TreasuryChanged(_treasury);
    }
```

Severity

Medium

Status

Fixed

[M-02] Missing zero check in recipient

in _transfer function there is not check recipient address . if mistakly sender send funds in zer address then all funds will be loss

add this

` require(recipient != address(0), "Invalid recipient address"); // Check for valid recipient

`

Severity

Medium

Status

Fixed

[L-01] Unused library should be remove

Remove Unused library

import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";

Severity

Low

Status

Fixed

Informative

Recommended to use Safemath library to avoid any calculation issue which lead overflow or underflow risk

Status

Acknowledge