



SMART CONTRACT SECURITY AUDIT

AXIR

Upgradeable Token Contract

CLIENT

AXIR

LANGUAGE

Solidity

REVIEW

September 2023

DOCUMENT

Document information

Project	AXIR, Upgradeable Token Contract
Client	AXIR
Lead auditor	Piyush Shukla
Platform	Not recorded in the original report
Language	Solidity
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Report date	September 2023

Document history

VERSION	DATE	CHANGE
1.0	September 2023	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	H-01 Centralization Risk : If owner Account is compromised then it make huge disaster to contract	8
	M-01 whenNotPaused modifier always be first modifier	9
	M-02 Missing zero check in recipient	9
	L-01 Unused library should be remove	9
	I-01 Recommended to use Safemath library	10
	Disclaimer	11

Executive summary

About AXIR

AXIR's UpgradeableAxirToken is an upgradeable, pausable token contract in which the owner mints and burns tokens up to a maximum supply.

Summary of the review

Safe Edges reviewed AXIR's upgradeable token contract. Five issues were identified: one High, two Medium, one Low and one Informational. The High issue is centralization risk: the owner alone can mint and burn tokens, so a compromised owner account controls the whole supply. AXIR acknowledged the High and Informational issues and fixed the other three.

Findings by severity



Findings by status

STATUS	ISSUES
● Acknowledged	2
● Resolved	3

Scope

Repository	Not recorded in the original report
Commit	Not recorded in the original report
Platform	Not recorded in the original report
Language	Solidity

IN-SCOPE FILES

UpgradeableAxirToken

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Not recorded in the original report.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
H-01	Centralization Risk : If owner Account is compromised then it make huge disaster to contract	High	● Acknowledged
M-01	whenNotPaused modifier always be first modifier	Medium	● Resolved
M-02	Missing zero check in recipient	Medium	● Resolved
L-01	Unused library should be remove	Low	● Resolved
I-01	Recommended to use Safemath library	Informational	● Acknowledged

Detailed findings

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

Severity **High** Status ● Acknowledged

DESCRIPTION

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 `onlyMinter` {
    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.

M-01 whenNotPaused modifier always be first modifier

Severity **Medium** Status ● Resolved

DESCRIPTION

Always whenNotPaused modifier to implement before any other modifier
correct the code in every function

RECOMMENDATION

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);  
}
```

M-02 Missing zero check in recipient

Severity **Medium** Status ● Resolved

DESCRIPTION

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

RECOMMENDATION

add this

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

L-01 Unused library should be remove

Severity **Low** Status ● Resolved

DESCRIPTION

Remove Unused library

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

I-01 Recommended to use Safemath library

Severity Informational Status ● Acknowledged

DESCRIPTION

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

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS