MONETHIC

Acurast

Smart Contract Audit

Prepared for:
Acurast Association

Date:
22.01.2026

Version:
Final, 1.0, for public release

Table of Contents

About Monethic

Monethic is a young and thriving cybersecurity company with extensive experience in various fields, including Smart Contracts, Blockchain protocols (layer 0/1/2), wallets and off-chain components audits, as well as traditional security research, starting from penetration testing services, ending at Red Team campaigns. Our team of cybersecurity experts includes experienced blockchain auditors, penetration testers, and security researchers with a deep understanding of the security risks and challenges in the rapidly evolving IT landscape. We work with a wide range of clients, including fintechs, blockchain startups, decentralized finance (DeFi) platforms, and established enterprises, to provide comprehensive security assessments that help mitigate the risks of cyberattacks, data breaches, and financial losses.

At **Monethic**, we take a collaborative approach to security assessments, working closely with our clients to understand their specific needs and tailor our assessments accordingly. Our goal is to provide actionable recommendations and insights that help our clients make informed decisions about their security posture, while minimizing the risk of security incidents and financial losses.

About Project

Acurast is a decentralized physical infrastructure network (DePIN) that transforms smartphones into a global, secure, and confidential edge computing network.

AcuERC20 is an ERC20 token with features such as EIP-2612 permit and a configurable transfer-restriction hook to a standard ERC-20.

AcurastToken inherits it, fixes 12 decimals, and mints the initial balances at deploy.

Disclaimer

This report reflects a rigorous security assessment conducted on the specified product, utilizing industry-leading methodologies. While the service was carried out with the utmost care and proficiency, it is essential to recognize that no security verification can guarantee 100% immunity from vulnerabilities or risks.

Security is a dynamic and ever-evolving field. Even with substantial expertise, it is impossible to predict or uncover all future vulnerabilities. Regular and varied security assessments should be performed throughout the code development lifecycle, and engaging different auditors is advisable to obtain a more robust security posture.

This assessment is limited to the defined scope and does not encompass parts of the system or third-party components not explicitly included. It does not provide legal assurance of compliance with regulations or standards, and the client remains responsible for implementing recommendations and continuous security practices.

Scoping Details

The purpose of the assessment was to conduct an audit against Acurast Smart Contracts, shared with Monethic as a deployed token address.

Scope

The scope of the assessment includes the smart contracts deployments listed below:

- `https://etherscan.io/address/0x216b3643ff8b7BB30d8A48E9F1BD550126202AdD#code`
    - `AcurastToken.sol`
    - `AcuERC20.sol`

Timeframe

On 01.10.2025 Monethic was requested for two Acurast Solidity Smart Contracts audits. Work began immediately.

On 02.10.2025, the report from the Smart Contract security assessment was delivered to the Customer.

On 22.01.2026, the Final report from the Smart Contract security assessment was delivered to the Customer.

Vulnerability Classification

All vulnerabilities described in the report were thoroughly classified in terms of the risk they generate in relation to the security of the contract implementation. Depending on where they occur, their rating can be estimated on the basis of different methodologies.

In most cases, the estimation is done by summarizing the impact of the vulnerability and its likelihood of occurrence. The table below presents a simplified risk determination model for individual calculations.

		Impact		
	Severity	**High**	**Medium**	**Low**
Likelihood	**High**	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Vulnerabilities that do not have a direct security impact, but may affect overall code quality, as well as open doors for other potential vulnerabilities, are classified as **Informational**.

Vulnerabilities summary

No.	Severity	Name	Status
1	Medium	Blacklist does not apply to spender in transferFrom	Acknowledged
2	Low	Approvals remain enabled while transfers are restricted	Acknowledged
3	Informational	No public getter for current restrictor address	Acknowledged
4	Informational	Duplicate entries in initial balances may mint multiple times	Acknowledged
5	Informational	Ambiguous restriction state	Acknowledged

Technical summary

1. Blacklist does not apply to spender in `transferFrom`

Severity: Medium

Location

- `AcuERC20.sol`

Description

The restrictor is meant to block blacklisted actors from moving tokens, but the `restrictTransfer(from, to, value)` check only considers `from` and `to` and ignores the actual caller. Due to this, a blacklisted account can still drain any previously granted allowances by routing tokens from an allowed owner to any allowed recipient it controls.

```
    modifier restrictTransfer(address from, address to, uint256 value) {
        if (!transfersAreUnrestricted) { // for efficiency we skip if zero address is set
            if (!IERC20TransferRestrictor(erc20TransferRestrictor).isTransferAllowed(from, to, value)) {
                revert IERC20TransferRestrictor.TransferRestricted();
            }
        }
        _;
    }
```

Remediation

In `transferFrom`, also validate the spender by extending the restrictor to `isTransferAllowed(from, to, spender, value)` or add a second check against `msg.sender` and deny when the caller is blacklisted.

Status: Acknowledged

2. Approvals remain enabled while transfers are restricted

Severity: Low

Location

- `AcuERC20.sol`

Description

The system aims to block value movement during restriction, but only `transfer/transferFrom` are affected while `approve/permit` remain open, letting users pre-stage allowances and signatures that execute immediately once restrictions are lifted. Those functions are part of inherited contracts hence still are callable.

```
abstract contract AcuERC20 is ERC20, ERC20Permit, ERC20Bridgeable,
AccessControlDefaultAdminRules
```

Remediation

We recommend restricting `approve` and `permit` with the same requirements by overriding them.

Status: Acknowledged

3. No public getter for current restrictor address

Severity: Informational

Location

- `AcuERC20.sol`

Description

Operations should be transparent, but the restrictor address is `private` and only revealed via events or storage reads, making on-chain inspection difficult.

```
    address private erc20TransferRestrictor =
address(0x000000000000000000000000000000000000dEaD); // transfers are disabled
to start, can be extended with allowlists in future.
    bool private transfersAreUnrestricted = false;
```

Remediation

We recommend adding a public view getter for `erc20TransferRestrictor` and optionally for the unrestricted flag.

Status: Acknowledged

4. Duplicate entries in initial balances may mint multiple times

Severity: Informational

Location

- `AcurastToken.sol`

Description

The constructor mints per entry in the distribution list without deduplication, so repeating an address issues multiple mints to the same recipient. However this requires admin error.

Remediation

If uniqueness is required, enforce it off-chain or add a pre-check for duplicate addresses before minting.

Status: Acknowledged

5. Ambiguous restriction state

Severity: Informational

Location

- `AcuERC20.sol`

Description

The condition is represented by both the restrictor address and a separate boolean flag, which are updated together only in one function, so any future write to one without the other can accidentally bypass restrictions.

In other words, this is more error prone than for instance relying only on a single source of truth like only the contract or only a variable.

```
    modifier restrictTransfer(address from, address to, uint256 value) {
        if (!transfersAreUnrestricted) { // for efficiency we skip if zero
address is set
            if
(!IERC20TransferRestrictor(erc20TransferRestrictor).isTransferAllowed(from, to,
value)) {
                revert IERC20TransferRestrictor.TransferRestricted();
            }
        }
        _;
    }
```

Remediation

We recommend deriving the condition directly from a contract or a variable.

Status: Acknowledged

END OF THE REPORT