Acurast Token

Acurast Association

Independent security assessment report

inference

Report version: 1.0 / date: 02.10.2025

inference

Table of contents

Table of contents **2**
Summary **4**
Overview on issues and observations 4
Project overview **6**
Scope 6
Smart contract security assessment 6
Scope limitations 6
Methodology 7
Objectives 7
Activities 7
Security issues **8**
Observations **9**
O-ACU-001: Lack of validation on TransferRestrictor address updates 9
O-ACU-002: Unnecessary deployment costs 10
O-ACU-003: Lack of restriction status update 11
Disclaimer **12**
Appendix **13**
Adversarial scenarios 13
Risk rating definition for smart contracts 14
Glossary 15

Version / Date	Description
1.0 / 02.10.2025	Final version

Summary

Inference AG was engaged by the Acurast Association to perform an independent security assessment of the Acurast Token smart contract, an ERC20-based token contract.

Inference AG performed the security assessment based on the agreed scope, following our approach and activities as outlined in the "Project overview" chapter between the 1st of October 2025 and the 2nd of October 2025. Feedback from the Acurast team was received and Inference performed a reassessment.

Based on our scope and the activities we performed, our security assessment didn't reveal any security issues, but several observations, which, if addressed with appropriate actions, may improve the quality of Acurast Token. During our evaluation, the Acurast team provided answers to all raised observations.

This report only shows remaining open or partly resolved observations.

Overview on issues and observations

At Inference AG we separate the findings that we identify in our security assessments in two categories:

- Security issues represent risks to either users of the platform, owners of the contract, the environment of the blockchain, or one or more of these. For example, the possibility to steal funds from the contract, or to lock them in the contract, or to set the contract in a state that renders it unusable are all potential security issues;
- Observations represent opportunities to create a better performing contract, saving gas fees, integrating more efficiently into the existing environment, and creating a better user experience overall. For example, code optimizations that save execution time (and thus gas fees), better compliance to existing standards, and following secure coding best practices are all examples of observations.

Details for each reported issue or observation can be obtained from the "Security issues" and "Observations" sections.

	Severity / Status
Security issues	
There are no open known security issues.	
Observations	
O-ACU-001: Lack of validation on TransferRestrictor address updates	- / open
O-ACU-002: Unnecessary deployment costs	- / open
O-ACU-003: Lack of restriction status update	- / open

Project overview

Scope

Smart contract security assessment

The scope of the smart contract security assessment was the following smart contract:

- AcurastToken
  - /contracts/token/AcurastToken.sol, including the imported AcuERC20.sol

The files in scope were made available via a source code repo: https://gitlab.papers.tech/papers/acurast/hyperdrive-ethereum and our initial security assessment considered commit "aaba511c02b20220a7edcbf571a5047636d2b1ba"[1].

Scope limitations

Our security assessment is based on the following key assumptions and scope limitations:

- Any potential adversarial activities conducted by the administrator of the contract or operational errors by administrators were out of scope.
- The RESTRICTOR_UPDATER holder is assumed to be a trusted actor: adversarial activities and operational errors by this role were out of scope.
- Deployment and initial configuration of the deployed smart contract was out of scope.
- The key management of associated secret keys has not been assessed.
- The entities owning privileged roles have not been reviewed, assessed, or vetted in any form.
- The imported OpenZeppelin libraries have not been assessed. However, the correct usage has been reviewed.

[1] The sha256sum hash of the repository's ".zip" file is:
b8a2622329163eb70d4c1b902e3763e1229b0aac6dd31a77d2b6ef14c0840cc3

Methodology

Inference's methodology for security assessments comprises a source code review in the high-level language, followed by multiple rounds of Q&A with the development team to discuss findings and critical points that emerged during the first assessment. This process is iterated until a version where no new findings emerge is assessed.

In order to ensure a high quality in our security assessments, Inference is using subject matter experts having a high adversarial scenario mindset to spot potential issues in protocols under review. Additionally, for smart contract security reviews, we apply checklists derived from good practices and commonly known issues to document our work and ensure good coverage.

Furthermore, Inference maintains regular communications with the development team to ensure a correct understanding of the solution and environment, but also to make teams aware of any observations as soon as possible.

Inference's internal quality assurance procedures ensure that results of security assessments are challenged for completeness and appropriateness by a second independent expert.

Objectives

The objectives are the identification of security issues with regards to the assessed smart contracts and their conceptual design and specification. The security assessment also focuses on adversarial scenarios on specific use cases which have been listed in appendix "Adversarial scenarios". These were identified together with the Acurast team and checked during our security assessment.

Activities

Our security assessment activities for the defined scope were:

- Source code review of smart contract code written in Solidity

Our activities for the reassessment were:

- Reviewing and assessing the feedback received from the Acurast team.
- Reassessing security issues and observations from initial assessment in case they are claimed to be resolved

Security issues

There are no open known security issues.

Observations

O-ACU-001: Lack of validation on TransferRestrictor address updates

The function updateErc20TransferRestrictor accepts any address without validation (i.e., whether it is an EOA or implements the necessary interface).
While it is possible for the RESTRICTOR_UPDATER to change the address in case of issues, checking during an update would prevent potential temporary disruptions to the token's correct functioning.

Recommendation:
We suggest implementing validation controls when updating the TransferRestrictor address.

Comment from Acurast team:
The idea is that the restrictor was only used for early delivery tokens that are not transferable. After the tokens become transferable, we will have no more need of the restrictor flow and it should use as little gas as possible.

Reassessment:
We have decided to classify this issue as an observation as the RESTRICTOR_UPDATER has the permission to solve any temporary disruption by updating the address again.

O-ACU-002: Unnecessary deployment costs

The AcurastToken.sol constructor contains two potential improvements to reduce gas consumption.

First of all, it performs the following check for every item in the _initialBalances array:
`require(_initialBalances[i].amount <= type(uint256).max / multiplier, "Amount too large")`

However:

- amount is of type uint128
- multiplier is exactly 10^12, which is roughly 2^40 (using the approximation that 2^10 is circa 10^3)
- type(uint256).max / 10^12 ≈ 2^256 / 2^40 ≈ 2^216
- The uint128 amount is, at most, 2^128, which is ALWAYS less than 2^216, so this condition can never fail.

This additional check, performed in a loop, can significantly increase the deployment cost of the AcurastToken contract, as it will be repeated for each entry of the _initialBalances array.

The second improvement is to store the length of the _initialBalances array in a variable, avoiding its recomputation at every iteration of the loop.

Recommendation:
We suggest removing the check, as it serves no purpose. We also recommend optimizing for gas savings by storing the array length in a variable.

Comment from Acurast team:
The Acurast Association Team acknowledges the input, but as the check is not harmful it was not removed for the productive deployment.

Reassessment:
We have decided to classify this issue as an observation, as we have received information from the Acurast team that the size of the _initialBalances array is small, thus the additional gas costs of this redundant check and the loop optimization are limited.

O-ACU-003: Lack of restriction status update

When the “erc20TransferRestrictor” address is updated, the “transfersAreUnrestricted” flag may change implicitly if the new address is “address(0)”.
While an “ERC20TransferRestrictorContractUpdated” event is emitted, it does not explicitly state that all transfers have become unrestricted. It would be advisable to emit a specific event for this scenario, for off-chain monitoring tools to track the token's transferability status.

Recommendation:
We recommend creating a new event and emitting it in case the transfer restriction status changes.

Comment from Acurast team:
The Acurast Association Team acknowledges the input, but as it's expected that the restrictor is updated a single time and then stays on "unrestricted" no courtesy change will be applied.

Reassessment:
We have decided to classify this issue as an observation as it does not pose a security risk.

Disclaimer

This security assessment report (“Report”) by Inference AG (“Inference”) is solely intended for the “Client” with respect to the Report’s purpose as agreed by the Client. The Report may not be relied upon by any other party than the Client and may only be distributed to a third party or published with the Client’s consent. If the Report is published or distributed by the Client or Inference (with the Client’s approval) then it is for information purposes only and Inference does not accept or assume any responsibility or liability for any other purpose or to any other party.

Security assessments of a software or technology cannot uncover all existing vulnerabilities. Even an assessment in which no weaknesses are found is not a guarantee of a secure system. Generally, code assessments enable the discovery of vulnerabilities that were overlooked during development and show areas where additional security measures are necessary. Within the Client’s defined time frame and engagement, Inference has performed an assessment in order to discover as many vulnerabilities of the technology or software analysed as possible. The focus of the Report’s security assessment was limited to the general items and code parts defined by the Client. The assessment shall reduce risks for the Client but in no way claims any guarantee of security or functionality of the technology or software that Inference agreed to assess. As a result, the Report does not provide any warranty or guarantee regarding the defect-free or vulnerability-free nature of the technology or software analysed.

In addition, the Report only addresses the issues of the system and software at the time the Report was produced. The Client should be aware that blockchain technology and cryptographic assets present a high level of ongoing risk. Given the fact that inherent limitations, errors or failures in any software development process and software product exist, it is possible that even major failures or malfunctions remain undetected by the Report. Inference did not assess the underlying third party infrastructure which adds further risks. Inference relied on the correct performance and execution of the included third party technology itself.

Appendix

Adversarial scenarios

The following adversarial scenarios have been identified and checked during our security assessment.

Scenario	Assessment result
As a normal user, add myself as an owner.	**Ok** Nothing identified.
As a normal user, execute restricted functionality.	**Ok** Nothing identified.
Exploit improper accounting to mint more tokens.	**Ok** Nothing identified.

Risk rating definition for smart contracts

Severities are quantified with two dimensions, roughly defined as follows, whereas the examples have to be regarded as indication only:

Probability of occurrence / materialisation of an issue
(bullets for a category are linked with each other with “and/or” condition.)

- Low:
  - A trusted / privileged role is required.
  - Contract may end up in the issue if other conditions, which are also unlikely to happen, are required.
- Medium:
  - A specific role or contract state is required to trigger the issue.
  - Contract may end up in the issue if another condition is fulfilled as well.
- High:
  - Anybody can trigger the issue.
  - Contract’s state will over the short or long term end up in the issue.

Impact:
(bullets for a category are linked with each other with “and/or” condition.)

- Low:
  - Non-compliance with standards
  - Unclear error messages
  - Confusing structures
- Medium:
  - A minor amount of assets can be withdrawn or destroyed.
- High:
  - Not inline with the specification
  - A non-minor amount of assets can be withdrawn or destroyed.
  - Entire or part of the contract becomes unusable.

Severity:

	Low impact	Medium impact	High impact
High probability	**High**	**Critical**	**Critical**
Medium probability	**Medium**	**High**	**Critical**
Low probability	**Low**	**Medium**	**High**

Glossary

Term	Description
ERC20	Technical standard for fungible tokens on the Ethereum blockchain.