stXTZ for Ubinetic

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 21.10.2025

inference

Table of contents

Version / Date	Description
1.0 / 21.10.2025	Final version

Summary

Inference AG was engaged by the Tezos Foundation (hereinafter "TF") to perform an independent security assessment of Ubinetic's stXTZ project.
stXTZ[1] is a derivative token representing user-staked XTZ, enabling users to maintain full liquidity of their staked tokens. It is powered by the Acurast decentralized compute network solution and governed through a DAO.

Inference AG performed the security assessment based on the agreed scope, following our approach and activities as outlined in the "Project overview" chapter. Feedback from the Ubinetic team was received, and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our security assessment revealed several security issues rated from high to low severity. Additionally, different observations were also made, which, if resolved with appropriate actions, may improve the quality and users' understanding of stXTZ.

This report only shows remaining open or partly resolved issues and 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.

[1] https://docs.youves.com/stxtz/stXTZ/

	Severity / Status
Security issues	
S-UST-001: Loss or compromise of manager key	high / open
S-UST-002: FA2 non-compliance - 0 token transfers	low / open
S-UST-003: FA2 non-compliance - non-existing tokens	low / open
S-UST-004: Depletion of gas station	low / open
Observations	
O-UST-001: Inherited Tezos staking model	- / open
O-UST-002: Change of min_amount	- / open
O-UST-003: Change of Tezos protocol parameters	- / open
O-UST-004: Uneven distribution of slashing penalties	- / open
O-UST-005: Inaccurate slashing risk until claim_outstanding invocation	- / open
O-UST-006: Inaccurate price recording during withdrawal	- / open
O-UST-007: No two-step procedure to replace admin	- / open
O-UST-008: Missing last-admin safeguard	- / open

Project overview

Scope

Smart contract security assessment

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

- Gas station: contracts/tracker/gas_station/gas_station.spy
- stXTZ token: contracts/tracker/stxtz_pool/stxtz_token.spy
- stXTZ pool: contracts/tracker/stxtz_pool/stxtz_pool.spy

The files in scope were made available via a source code repo: https://gitlab.papers.tech/papers/ubinetic/sap-smart-contract and our initial security assessment considered commit “0eed9a57909a3e15c75a903c98e7e56e7eba3f08”.

Our reassessment considered commit “0eed9a57909a3e15c75a903c98e7e56e7eba3f08”.

Acurast scripts security assessment

Also in scope for the security assessment were the following Acurast scripts:

- src/index.ts
- src/manager.ts
- src/types.ts
- src/utils.ts
- src/certificates.ts
- src/keyManager.ts

The files in scope were made available via a source code repo: https://gitlab.papers.tech/papers/ubinetic/stxtz/stxtz-acurast-scripts and our initial security assessment considered commit “d96087afaf4734876f4167ec4945d69673d777d4”.

Our reassessment considered commit “28f7af04599c9dbb86131b52a7df1bbf0a1b77fa”.

Scope limitations

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

- Adversarial actions or operational errors by privileged contract roles were not within scope.
- Deployment procedures and initial configuration of the smart contract were not reviewed.
- The stXTZ solution was analyzed under the assumption that only a single manager exists.
- The potential impact of Acurast devices failing to execute in a timely manner was not assessed.
- The Acurast protocol, platform, and device management processes were excluded from this review.
- Vetting of Acurast device holders, including the security of their storage and handling practices, was not conducted.
- Claims regarding decentralization, including the DAO implementation and associated privileged roles, were not reviewed, assessed, or validated.

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.

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 based on the Tezos smart contract security assessment checklist[2] to document our work and to ensure good issue 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 the results of security assessments are challenged for completeness and appropriateness by a second independent expert.

[2] https://github.com/InferenceAG/TezosSecurityAssessmentChecklist/tree/master/publications/v2.0

Objectives

The objectives are the identification of security issues with regards to the assessed smart contracts and Acurast scripts 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 Ubinetic team and checked during our security assessment.

Activities

The security review was conducted by two independent auditors, with a total of 7.5 person-days allocated to the assessment.

Our security assessment activities for the defined scope were:

- Source code review of smart contract code written in SmartPy
- Security assessment of the Acurast scripts

Our activities for the follow-up assessment were:

- Source code review of the changes applied to the smart contract code written in Michelson
- Source code review of changes in the Acurast scripts
- Reassessing security issues and observations from the initial assessment in case they are claimed to be resolved

Security issues

S-UST-001: Manager key handling risks

The Acurast scripts implement a key management process in which a manager key is generated and distributed across all participating Acurast devices for storage. At present, 39 devices are registered in the relevant smart contract (KT198mB5VXfeftAXbo3dqkEMjR3oWShEoazY), with one device designated as the "leader."

While the Acurast protocol, platform, and device management processes were not included in the scope of this assessment, several risks arise from the chosen key distribution and storage model:

- Key loss and regeneration by the leader
  If the leader device loses its copy of the manager key and cannot communicate with other Acurast devices (e.g., due to loss of Internet connectivity), the leader may generate a new manager key.
  This behavior stems from the execution of this.pendingResponses.delete(peerId) within the requestKeyFromOtherProcessors function, which leads to key regeneration when peer communication fails.
- Unintended propagation of a new manager key
  A regenerated manager key may remain isolated if other devices do not also lose their keys. However, in scenarios where another device requests a key and the leader responds first, the newly generated key may propagate.
  Additionally, if a device intentionally or mistakenly sends unsolicited key responses, other devices could overwrite their local copy of the manager key, causing the new key to spread network-wide.
- Simultaneous key loss across devices
  All devices could lose their manager key due to software updates, system upgrades, vulnerabilities, or bugs in the Acurast platform/protocol.
  Such a scenario would leave the network unable to recover the original key, potentially triggering propagation of a new manager key.
- Risk of key extraction from devices
  The more tez controlled by the manager key, the greater the incentive for adversaries to attempt extraction.
  Attack vectors include side-channel attacks, hardware bus snooping, and forensic analysis, all made possible through direct physical access to Acurast devices.

Probability - Low.
Impact - High. In the event of key compromise or widespread key loss, all tez controlled by the manager key could be irreversibly lost.
Severity - High.

Recommendation:
We recommend considering the following non-exhaustive list of suggestions[3]:

- Leader role mitigation:
  Avoid assigning a persistent “leader” role after the manager key has been generated, to prevent automatic regeneration of a new manager key. Remove the “leader” flag in KT198mB5VXfeftAXbo3dqkEMjR3oWShEoazY.
- Validation of key responses:
  Implement checks to ensure that only solicited key responses are accepted. For instance, validate the result of this.pendingResponses.delete(peerId) in the onKeyResponse function of keyManager.ts.
- Isolate Acurast devices
  Keep a subset of Acurast devices offline to maintain independent copies of the manager key, ensuring recovery in the event of widespread loss.
- Segmentation of custody
  Operate multiple, strictly separated Acurast stzXTZ clusters, each responsible for a subset of tez, thereby limiting the impact of a single key compromise or propagation failure.
- Controlled update strategy
  Apply staged updates (e.g., update rings) to reduce the risk of simultaneous failure across all devices.

[3] Note: Some of the suggested measures (e.g., staged update rings) may already be in place. However, this could not be verified, as Acurast device management was outside the scope of this assessment.

S-UST-002: FA2 non-compliance - 0 token transfers

The stXTZ token smart contract fails if a transfer of zero (“0”) stXTZ tokens is initiated from an address that is not yet registered in the ledger.

According to the FA2 standard (TZIP-012), in the section “Core Transfer Behavior”:

- If the token owner does not hold any tokens of type `token_id`, the owner's balance is interpreted as zero. No token owner can have a negative balance (TZIP-012).
- Transfers of zero amount MUST be treated as normal transfers (TZIP-012).

Based on these requirements, Inference interprets that such zero-amount transfers should not fail.

The current behavior introduces a risk that certain transfers could fail in specific situations or for solutions interacting with the stXTZ token smart contract, potentially leading to unexpected errors or incompatibilities.

Probability - Low.
Impact - Low.
Severity - Low.

Recommendation:
We recommend analysing the situation and potentially adapting the stXTZ token smart contract in order to be compliant with the FA2 standard (TZIP-012)

S-UST-003: FA2 non-compliance - non-existing tokens

According to the FA2 standard (TZIP-012), in the section “Core Transfer Behavior”:

- If one of the specified `token_id`s is not defined within the FA2 contract, the entrypoint MUST fail with the error mnemonic `"FA2_TOKEN_UNDEFINED"`. (TZIP-012).

In the stXTZ contract, the transfer entrypoint does fail when an undefined token_id is used, but it returns the incorrect error mnemonic "FA2_INSUFFICIENT_BALANCE". Additionally, the update_operators entrypoint does not fail at all when provided with a non-existent token_id.

This non-compliant behavior introduces a risk of unexpected behavior, errors, or incompatibilities when interacting with the stXTZ token smart contract.

Probability - Low.
Impact - Low.
Severity - Low.

Recommendation:
We recommend analysing the situation and potentially adapting the stXTZ token smart contract in order to be compliant with the FA2 standard (TZIP-012).

S-UST-004: Depletion of gas station

Currently, gas cost estimations are obtained from a single RPC provider. If this RPC provider were compromised or manipulated by the baker, there is a potential risk that the baker could deplete the gas station contract.

Probability - Low.
Impact - Low.
Severity - Low.

Recommendation:
Consider querying multiple RPC providers for gas estimations to reduce reliance on a single source and further mitigate this risk.

Observations

O-UST-001: Inherited Tezos staking model

The stXTZ solution resembles the staker model of the Tezos protocol and therefore inherits the following characteristic from it.
The distribution of rewards by the protocol in “Adaptive Issuance”, when the “edge-of-baking-over-staking” parameter is less than “1,” does not properly reflect the staker’s contribution to the baker’s rights:

1. When a staker stakes with a baker, the staker immediately begins receiving rewards in the next cycle. However, the staker’s stake only impacts the baker’s rights in the cycle “current cycle + 1 + CONSENSUS_RIGHTS_DELAY.”
2. When a staker unstakes, they immediately stop receiving rewards. Thus, potential rewards for the ongoing cycle and the following two cycles, where the staker’s stake contributed to the baker’s rights, are not credited to the staker.

This rewards distribution does not compromise the security of the blockchain or the baker. However, it can be seen as an undocumented fee to the baker and it slightly disadvantages large stakers.

Scenario illustrating a “hidden fee” to the baker paid by the staker:

1. A baker has a stake of 6,000.
2. A large staker adds a stake of 5 x 6,000 (no overstaking).
3. The large staker then unstakes everything.

In this scenario, the large staker receives staking rewards based on a baking power of 6,000 for a few cycles, instead of the rewards for a baking power of 30,000.

Another scenario occurs when, after step 3 in the above scenario, a small staker with 6,000 immediately starts staking (or at least before the first baking and before the cycle ends). The rewards (“hidden fee” for large staker) from the cycle with a baking power of 30,000 will then be shared between the baker and the small staker.

The latter scenario could yield more rewards for a small staker than staying with the same baker if the small staker unstakes again after collecting the rewards from the 30,000 baking power cycles. However, this strategy is only beneficial if the "jumping" staker timely and

consistently finds a suitable spot to take over each time its unstaked assets can be finalized.

We flagged this point not as a security issue, but as an observation to make users aware of how the Tezos protocol behaves, which is inherited by the stXTZ solution.

Recommendation:
We recommend informing users about how stXTZ works, including the underlying Tezos mechanisms it inherits.

O-UST-002: Change of min_amount

When administrators increase the “min_amount” parameter in the stXTZ pool contract, certain contract entrypoints may become temporarily inaccessible under specific conditions:

- The claim entrypoint cannot be executed by managers if there is only a single pending claim with a previously deposited amount lower than the newly set min_amount. Although this situation may automatically resolve once the accumulated claimable amount exceeds the updated min_amount, the affected user can alternatively invoke cancel_deposit.
- The pending_withdrawal entrypoint cannot be executed by managers if there is only a single pending withdrawal request with an amount lower than the newly set min_amount. This situation may resolve automatically once the total amount requested meets the updated threshold; alternatively, the user can invoke cancel_request_withdrawal as a fallback.

While these situations may self-correct or be mitigated by cancellation options, they introduce temporary usability issues that could affect user experience and contract reliability.

We have classified this as an observation rather than a security issue, as it does not constitute a protocol vulnerability per se. The current distribution mechanism does not disadvantage the stXTZ solution itself.

Recommendation:
We recommend analyzing this situation and developing a design that better aligns stXTZ with the actual Tezos protocol.

O-UST-003: Change of Tezos protocol parameters

The stXTZ pool contract administrators can modify the parameters blocks_per_cycle and blocks_offset, with changes taking effect immediately.
It’s crucial that these modifications occur simultaneously with the Tezos protocol's updates during a protocol upgrade, or the stXTZ solution may incorrectly allow users to claim their tez either too early or too late. Consequently, users might be able to invoke the claim_outstanding entrypoint prematurely or, conversely, be prevented from doing so when they should already be eligible.

If other protocol constants—such as CONSENSUS_RIGHTS_DELAY—change, the stXTZ pool contract would need to be replaced, as there are no functions to update these protocol-dependent parameters.

We have classified this as an observation rather than a security issue, as it depends on the specifics of future protocol upgrades and on potential delays or misaligned actions by the stXTZ solution administrators.

Recommendation:
We recommend closely monitoring changes in upcoming Tezos protocol upgrades, to assess the impact on the stXTZ solution, and to take appropriate actions timely.

O-UST-004: Uneven distribution of slashing penalties

The slashing penalty applied to bakers is not distributed evenly among users with tez at stake. Users whose funds are in the process of unstaking may incur reduced penalties or avoid them entirely.

For example, when a user initiates unstaking, the current stXTZ token price is recorded. If a baker subsequently commits a minor infraction, such as a “double attestation” for a small fraction of their attestation rights, the resulting slashing may be minimal, and the current stXTZ token price is decreased accordingly.

When the user later claims their finalized XTZ by invoking the “claim_outstanding” entrypoint, the amount to be received—initially calculated based on the recorded stXTZ token price—is adjusted only if the user’s recorded price is lower than the current stXTZ token price. As a result, the user may avoid paying any slashing penalty, if the stXTZ token price had already recovered to the user’s recorded price at the time of claiming.

Furthermore, the user may delay claiming their finalized XTZ until the stXTZ token price has recovered to their recorded price. However, doing so also exposes them to the risk of additional slashing penalties. See also observation O-UST-005: Inaccurate slashing risk until claim_outstanding invocation

This misalignment may result in unfair treatment of active stakers compared to those in the process of unstaking. Other variations of this issue may exist, potentially impacting the proportionality and integrity of penalty distribution.

We have classified this as an observation rather than a security issue, as it does not constitute a protocol vulnerability per se. The current distribution mechanism does not disadvantage the stXTZ solution itself.

Recommendation:
We recommend analyzing this situation and developing a design that better aligns stXTZ with the actual Tezos protocol.

Furthermore, consider designing the claim_unstaking entrypoint to be permissionless, allowing anyone—such as an automated bot—to initiate the claim_unstaking process on behalf of a user.

O-UST-005: Inaccurate slashing risk until claim_outstanding invocation

When the Tezos blockchain marks tez as finalizable, these tez are no longer slashable by the Tezos protocol itself. However, assets in the stXTZ solution remain exposed to slashing until the user invokes the claim_outstanding entrypoint in the stXTZ pool contract.

If an incident such as double-signing occurs after the user's funds have effectively been unstaked on a Tezos protocol level, the user's tez may still be slashed. This creates a period during which tez are technically no longer "at stake" but remain vulnerable to penalties—particularly if there is a delay before the user invokes claim_outstanding or if the Acurast system experiences processing delays or interruptions, preventing timely execution of the claim_outstanding entrypoint by users.

This misalignment between the actual staking state and the slashing logic could result in users being unfairly penalized even after their funds should no longer be at risk.

We have classified this as an observation rather than a security issue, as it does not constitute a vulnerability per se, but represents a risk scenario that potential users should be aware of.

Recommendation:
We recommend analyzing this situation and developing a design that better aligns stXTZ with the actual Tezos protocol.

At a minimum, users should be informed about this discrepancy so they can invoke claim_unstaking immediately once their tez are ready to be claimed.

Furthermore, consider designing the claim_unstaking entrypoint to be permissionless, allowing anyone—such as an automated bot—to initiate the claim_unstaking process on behalf of a user.

O-UST-006: Inaccurate price recording during withdrawal

When a user invokes the “request_withdrawal” entrypoint in the stXTZ pool contract, the current stXTZ token price is recorded in the withdrawal_queue for that user. However, this mechanism does not accurately reflect the user’s position. From the moment “request_withdrawal” is called until the Acurast script executes the actual unstaking operation on the blockchain, the user no longer accrues rewards, despite their stake remaining active.

This results in a misalignment where the user’s tez are still “at stake” but no longer generate rewards. The impact of this issue increases significantly if execution of the Acurast script is delayed, for example due to operational or network issues, potentially leading to a loss of expected yield for the user.

We have classified this as an observation rather than a security issue, as it does not constitute a vulnerability per se, but represents a risk scenario that potential users should be aware of.

Recommendation:
We recommend analyzing this situation and developing a design that better aligns stXTZ with the actual Tezos protocol.

At a minimum, users should be informed about this discrepancy so they are aware of the differences to the native Tezos protocol.

O-UST-007: No two-step procedure to replace admin

The set_administrator entrypoint enables the assignment of a new address to the administrator role in the stXTZ token smart contract. Once it is confirmed that the call originates from the current administrator, the contract unconditionally updates the administrator role to the provided address.

This mechanism poses a risk if the supplied address is incorrect, invalid, or inaccessible. Such a situation could result in governance issues, particularly if other administrators are subsequently removed, potentially leaving the contract without an effective administrator.

We have classified this as an observation only and not as a security issue, as adding and removing administrators can only be performed by administrators of the contract.

This observation affects the stXTZ token smart contract in scope only.

Recommendation:
We recommend implementing a two-step procedure to change any critical privileged addresses. In the first step the current privileged address proposes a new address, followed by a second step in which the newly proposed address accepts this proposal. The change of the critical privileged address is only done after the acceptance at the second step.

At the very least, we recommend analysing the situation, ensuring with appropriate documentation and procedures that admins dealing with these contracts are not mistakenly removing the last existing admin.

O-UST-008: Missing last-admin safeguard

The smart contracts do not implement any safeguard to ensure that at least one working “administrator” is always available. If all administrators of the smart contracts lose their privileges, either accidentally or on purpose, there is no alternative way to restore an administrator.

We have classified this as an observation only and not as a security issue, as changing the administrators can only be performed by administrators of the contracts.

This observation affects all smart contracts in scope.

Recommendation:
We recommend implementing a safeguard to ensure that the last remaining administrator cannot be removed, thereby preventing the contract from being left without an active administrator.

Disclaimer

This security assessment report (“Report”) by Inference AG (“Inference”) is solely intended for the Tezos Foundation (“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
External delegation / Overdelegation Baker has external delegation or is even overdelegated.	**Ok** Possible, but no incentive to do so, since no payout.
Overstaking Baker is overstaked.	**Ok** Not possible, since tez are directly staked by the baker. No external stakers.
Minimum baking requirements not achieved due to many external stakers In case of accepting external stakers the baker may have too few of their own tez staked in order to reach the minimum total of 6000 staked tez.	**Ok** No external stakers.
Draining of tez by consensus key The consensus key is able to drain significant funds from the baker's liquid tez.	**Ok** Nothing identified to drain a significant amount of funds. Neither in the "staking" nor "finalize unstaking" process. Also nothing identified with regards to the implicit "finalize unstake" when "unstake" or "stake" are called.
Double signing by consensus key - unintentionally The operator of the consensus key double signs leading to slashing.	**Note** This can technically not be prevented. However, the current setup will punish all managers and users/stakers.

Double signing by consensus key - intentionally The operator of the consensus key double signs and self denunciates, while denunciation is included in a block controlled by the operator. Thus, parts of the slashed amount are credited to the operator.	**Note** We have been informed that under the current model, which relies on a single baker, the baker is considered a trusted party.
Bad performance of a baker A baker frequently misses blocks (or ⅔ of the attestations in a cycle), resulting in not earning the maximum possible rewards for the assigned stake.	**Note** We have been informed that under the current model, which relies on a single baker, the baker is considered a trusted party.
Compromise or loss of consensus key The operator of the baker reports to Ubinetic that their consensus key[4] has been compromised or lost.	**Note** A new consensus key can be set. With the consensus key only a few tez can be stolen, since most tez are staked and liquid tez is regularly skimmed. However, double signing can technically not be prevented.
Compromise of manager key / device The manager key / device is compromised.	**Risk exists** See our raised point: S-UST-001: Manager key handling risks
Smart contract - As a normal user, add myself as an admin/owner. - As a normal user, execute admin/owner functionality. - As a normal user, execute functionality of defined roles. - Exploit improper accounting to obtain more stXTZ or tez. - Exploit improper accounting to obtain more rewards.	**Ok** Nothing identified.

[4] A consensus key only can perform the following: drain tez from manager key, sign pre-/attestation, and sign blocks. No other operations possible with the consensus key (e.g. “finalize unstake” is not possible).

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 TZIP 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
Acurast	Decentralized compute network solution. Website: https://acurast.com/
DAO	Decentralized Autonomous Organisation
SmartPy	High level smart contract language for the Tezos blockchain. Website: https://smartpy.tezos.com/