tzSafe Audit

LIGOLang

Revisions

v0: 2023-07-10

v1: 2023-09-07

v2: 2023-12-18 – updated to tzsafe 0.3.3

Scope

We reviewed the tzSafe Cameligo code, at revisions 98b6da5c12ff, 763e9274fec9, and cee65a3f27b0. We did not review the compiled Michelson.

We lightly reviewed the application UX and code as well (around revision 2773f86b279b) but did not consider it in scope.

Summary

The contract itself appears to be quite good. We found only one critical problem in the contract code, now resolved.

~~Contract can reach an "invalid" state~~	**Contract**, **Critical**, **Resolved**
~~Owners can trick other owners~~	**App**, **Vuln**, **Critical**, **Resolved**
Possible confusion about settings	**App**, **Confusion**
Confusion from concurrent settings changes	**App**, **Confusion**
Linear storage cost to transfer assets	**Confusion**
Linear storage cost in number of owners	**Confusion**
Users should be able to change their vote	**Contract**, **Confusion**
Possible client DoS by owners	**App**, **Confusion**

Legend:

Contract: an issue with the contract code itself. We only found one, and it is resolved!
App: an issue not with the contract code but only the application; fixable without redeploying multisig instances
Vuln: an issue which could be exploited by an attacker
Confusion: a possible risk of liveness or safety issues due to user confusion or ignorance

Considerations

Safety

The key safety concern is that if the contract emits an operation, it should have been intended by the owners.

Functionally, we consider an operation to be "intended by the owners" when enough owners have "signed" the operation, based on the configured threshold.

However, it is important to consider owners' real intent holistically over the span of multiple operations on the contract and in the context of the UI.

Liveness

The liveness concern is simpler: owners should retain the ability to cause the contract to emit operations, with constant gas/storage costs. Otherwise, value can be lost, e.g. in the form of stuck tez or token assets.

We mainly considered potential liveness problems in the contract itself. It may also be important to consider liveness problems in the client and infrastructure. One hopes that such problems could be fixed if/when they arise.

Trust

The key function of tzSafe is to distribute authority across owners, limiting the authority of each owner. Therefore, we do not trust individual owners completely, and we do consider possible attacks by owners. We do trust a quorum of owners (according to the configured threshold): e.g. if a quorum of owners *intentionally* lock up the contract forever, that is not a liveness issue but is working as intended.

Findings

~~Contract can reach an “invalid” state~~ (Critical, Resolved)

The contract code includes a settings validation function, which checks e.g. that the threshold is at least equal to the number of owners, among other things.

Previously, it was possible to enter an invalid state, even by using the official UI (e.g. by proposing multiple changes to owners and threshold and then executing them in an unfortunate order.)

The contract would then be locked forever as it would detect the invalid state and fail every operation immediately.

This has been fixed in revision 6fa25a1ddb8a by performing validation at the end of each contract execution. So, if executing a proposal would cause the contract to enter an invalid state, the execution will simply fail.

~~Owners can trick other owners~~ (Critical, Resolved)

Some proposals (including delegation changes and calls to other contracts) are represented with two pieces of data: a lambda containing the code which would be executed by the contract, and a piece of metadata (a JSON string) which can describe the intended meaning of the lambda.

Previously, the UI displayed the proposal to users by inspecting the metadata only. This meant that a single malicious owner with a little bit of technical skill could trick other owners, by submitting proposals with faked metadata, which did not accurately represent the lambda code. For example, it was possible that owners would be shown a “set delegate to baker X” proposal for signing, only to find on execution that the proposal actually steals all the tez in the contract.

This issue has been addressed in more recent versions of the tzSafe UI. We did not examine this new UI code very closely, but it appears to do the right thing.

A related issue is that even when the client correctly determines that a proposal lambda is unrecognized and the effects are arbitrary, the client might fail to appropriately warn signers about the possible effects, and may even allow the proposer to mislead signers by displaying metadata chosen by the proposer. **This issue also has been addressed in the most recent versions of tzSafe UI.**

Possible confusion about settings

Users could have problems if the “settings” (proposal duration, owners, threshold) are changed to bad values. Examples:

- The proposal duration could be set too short so that proposals expire before users are able to sign them. Since v0 of this audit, the UI has been updated to warn signers about this.
- The threshold could be set too high. In the worst case, one or more owners could be unable to sign (e.g. bogus or lost keys) and if the threshold is too high, the contract could become locked forever.
- The threshold could be set too low, allowing some few owners to take over.

Generally speaking, the UI does not explain the significance of these settings clearly, and does not seem to appropriately warn signers about the dangers. The UI for signing owner/threshold changes seems very confusing in particular. (See also next finding.)

It would not be very surprising for signers to get confused and accidentally set settings to bad values.

Confusion from concurrent settings changes

If multiple owner/threshold change proposals are pending, their effects can be confusing or even indeterminate.

Here is one arbitrary example. Suppose there are 50 owners and the threshold is 26/50. Two proposals are created concurrently:

1. add 10 owners, set threshold to 31/60
2. remove 10 (different) owners, set threshold to 21/40

Note that both proposals attempt to maintain a majority threshold. Suppose these proposals are both signed by at least 31 users. Now they can both be executed in either order, and the combined effects are both indeterminate and somewhat confusing. If proposal 2 is executed last, the final threshold will be 21/50. If proposal 1 is executed last, the final threshold will be 31/50. Both of these outcomes may be surprising to the owners: neither of these is a majority and neither outcome was actually chosen by proposers. In any case, it does not seem very plausible that owners *intended* for the outcome to be indeterminate, so this could be considered a safety issue.

It is not clear how to address this issue. It could suffice to “linearize” settings changes by enforcing in the client that owners must resolve any pending settings proposal before signing another one, but this could introduce liveness problems if owners are forced to wait for a pending proposal to expire. It might suffice to have some kind of warnings in the client, when signing a settings change while others are still pending.

We suggest that a redesign of the threshold mechanism should be considered for future versions of tzSafe. For example, it would seem to be much less confusing to configure the threshold by a rule like >50%, >2/3, etc, automatically keeping the required number of signers consistent with the number of owners. Other solutions are possible too. For example, a compare-and-swap-like approach could be used to enforce a linearization of settings changes without causing liveness issues, or each settings change proposal could describe the entire outcome. Such approaches would ensure that proposers and signers see the outcome they will actually get.

Linear storage cost to transfer assets

The design of the contract means that every proposal requires a storage burn proportional to the size of the proposal lambda. This means that transferring many unique assets (e.g. distinct FA2 tokens) away from the contract may be somewhat costly. Users should be aware of this cost before transferring many unique assets to the contract.

Linear storage cost in number of owners

Owners should be aware that the storage costs associated with each proposal are linear in the number of owners signing. So, there is also a theoretical maximum number of owners beyond which the contract will become inoperable.

We don’t expect owners to reach these limits in practice… Perhaps some reasonable maximum number of owners should be enforced in the client, though.

Users should be able to change their vote

The contract does not allow a user to change their vote on a proposal. Theoretically, this could lead to a safety/liveness dilemma in some situations.

Suppose there is a pending proposal and some users have already approved it, but they change their minds and create a modified proposal instead. Suppose they want *at most* *one* of these proposals to be executed. Because they cannot change their vote, and because the other owners might not cooperate, it is conceivable they may be forced to choose between waiting for the first proposal to expire (with the configured arbitrarily long proposal duration) before signing the second proposal, or taking a risk that both proposals might be executed by signing the second proposal while the first one is still pending. That is, they are forced to choose whether liveness or safety is violated.

We consider this a minor issue since such situations do not seem likely to arise in practice.

Possible client DoS by owners

The contract itself appears to be immune to DoS, but the client could potentially be DoS'd by a malicious owner. This could be addressed in the client, without changes to the contract.

Of course, there are many DoS risks not specific to tzSafe, inherent to the web and to Tezos. However, the tzSafe application specifically is designed to load and display *every* pending proposal. This seems to mean that one owner could DoS other owners by submitting very many proposals.

The worst case here is some very unusual situation, where one malicious owner stands to gain a lot from the contract being unusable for a certain period of time. One hypothetical scenario: if the tzSafe does a classic atomic swap and a malicious owner is on the other side, the malicious owner might reveal the secret while DoS'ing the tzSafe long enough to take both sides of the swap.

Of course, again, this is not specific to tzSafe; there is always a DoS risk with an atomic swap, for example.

However, tzSafe owners should either avoid such situations entirely, or use even longer deadlines than they normally would, leaving extra time for DoS problems specific to tzSafe to be mitigated in the application, or enough time to find a workaround (e.g. submitting transactions manually or using an alternate client.)