Technical Review of Gero Governance

Smart Contract Verification Team
July 7, 2022

Contents

Chapter 1

Executive Summary and Scope

This report is presented without warranty or guaranty of any type. This report lists the most salient concerns that have so far become apparent to Tweag after a partial inspection of the engineering work. Corrections, such as the cancellation of incorrectly reported issues, may arise. Therefore Tweag advises against making any business decision or other decision based on this report.

Tweag does not recommend for or against the use of any work or supplier referenced in this report. This report focuses on the technical implementation provided by the project's contractors and subcontractors, based on their information, and is not meant to assess the concept, mathematical validity, or business validity of MLabs's product. This report does not assess the implementation regarding financial viability nor suitability for any purpose.

Scope and Methodology

Tweag looks exclusively at the on-chain validation code provided by MLabs. This excludes all the frontend files and any problems contained therein. Tweag manually inspected the code contained in the respective files and attempted to locate potential problems in one of these categories:

a) Unclear or wrong specifications that might allow for fringe behavior.

b) Implementation that does not abide by its specification.

c) Vulnerabilities an attacker could exploit if the code were deployed as-is, including:

- race conditions or denial-of-service attacks blocking other users from using the contract,
- incorrect dust collection and arithmetic calculations (including due to overflow or underflow),
- incorrect minting, burning, locking, and allocation of tokens,
- authorization issues,

d) General code quality comments and minor issues that are not exploitable.

Where applicable, Tweag will provide a recommendation for addressing the relevant issue.

Chapter 2

Audit

2.1 Methodology

Tweag analysed the validator scripts comprising the Gero Governance protocol, contained in the repository[1] as of of commit `8f9f8520b121815979445696868 2d41a2265a951`. The names of the files considered in this audit and their `sha256sum` are listed in Table 2.1.

Our analysis is based on the documentation provided by MLabs, and on Slack conversations with MLabs. The relevant documentation files are listed in Table 2.2 and their contents will be referred to as *the specification* of the protocol.

In the beginning of the audit we were unable to interact with MLabs's contracts. Although the underlying cause is still unclear, we believe it to be a specific version of `plutus-apps`, which is a dependency of our auditing library `cooked-validators`. Tweag time boxed debugging this issue with MLabs but we were unable to find and fix the underlying cause. Therefore, together with MLabs, we decided to *downgrade* `cooked-validators` to a version that depended on a non-faulty `plutus-apps` and cherry-pick any necessary newer library features. For convenience, we are packaging this custom snapshot of `cooked-validators` in the vendor folder. Tweag *does not* commit to maintaining this custom version in the future.

2.2 Findings

Table 2.3 lists our concerns with the current Gero Governance implementation based on our partial exploration during a limited period of time. Throughout the rest of this section, we will detail each of our findings.

2.2.1 Vulnerabilities

2.2.1.1 ■ User tokens can be stolen by public keys

Severity: *Critical*

This vulnerability can no longer be triggered as of commit `a8ae4be99d3fbe8f6a60d43fb5db82c296a1fe81` of `github.com/mlabs-haskell/gero-gov`. We stress that this newer version of the code was not *audited. In particular, we have not investigated if the change introduced new vulnerabilities.*

When closing two user positions, each carrying its user token, on the same transaction, it is possible to burn only one of the tokens and pay the other to a public key. This is simple and cheap since an attacker only needs to open two positions, close them, and get one of the tokens back.

[1] `https://github.com/mlabs-haskell/gero-gov`

sha256sum	File Name
700f1ce...51196bc	gero-onchain/src/Gero/Onchain.hs
6910827...906b788	gero-onchain/src/Gero/Onchain/Compiled.hs
b45a006...fc3dd29	gero-onchain/src/Gero/Onchain/Compiled/Gov.hs
29c6081...e0abe05	gero-onchain/src/Gero/Onchain/Compiled/Mint.hs
36beb6e...7fae841	gero-onchain/src/Gero/Onchain/Compiled/UserStake.hs
bade97e...924ec55	gero-onchain/src/Gero/Onchain/Contracts.hs
b6abacf...7f0f486	gero-onchain/src/Gero/Onchain/Contracts/Gov.hs
69e7105...7f6b7bb	gero-onchain/src/Gero/Onchain/Contracts/Mint.hs
21d28ed...8df7c2b	gero-onchain/src/Gero/Onchain/Contracts/RawContext.hs
127e00c...ce67596	gero-onchain/src/Gero/Onchain/Contracts/UserStake.hs
72894d9...ac4b9e8	gero-onchain/src/Gero/Onchain/Contracts/Util.hs
77c8cbd...eddc8f4	gero-onchain/src/Gero/Onchain/Types.hs
7c53d49...c2f2bc2	gero-onchain/src/Gero/Onchain/Types/Common.hs
5f829a4...c5c6348	gero-onchain/src/Gero/Onchain/Types/Gov.hs
a3123c4...5df39cd	gero-onchain/src/Gero/Onchain/Types/Mint.hs
1198126...4165795	gero-onchain/src/Gero/Onchain/Types/UserStake.hs

Table 2.1: *On-chain code source files and their `sha256sum` that were analysed as part of the review*

sha256sum	File Name
3961900...c7b5ca3	docs/CHANGELOG.md
6506393...3e3720f	docs/gov-docs.md

Table 2.2: *Documentation files and their `sha256sum` that were used as the specification*

Severity	Section	Summary
Critical	2.2.1.1	User tokens can be stolen by public keys
High	2.2.4.1	Unclear role of **`Gov`** validator and token
Medium	2.2.1.2	The minting policy of the **`Gov`** token mostly relies on trusting the administrator
Medium	2.2.3.1	Error reporting is lacking
Low	2.2.2.1	User ids are not reused
Low	2.2.2.2	No specification on burning **`Gov`** tokens
Low	2.2.2.3	Unclear meaning of zero votes
Low	2.2.2.4	`gsPolicyIds` is an unused and immutable piece of datum
Low	2.2.3.2	Dead code
Low	2.2.3.3	Undocumented code section regarding closing positions
Lowest	2.2.2.5	Specification mentions unimplemented features
Lowest	2.2.2.6	Various minor concerns with the specification
Lowest	2.2.3.4	Various minor code quality concerns

TABLE 2.3: *Table of findings*

This double satisfaction attack is possible because there are two kinds of withdrawal transaction, depending on whether some stake remains in the user position or whether all stake is withdrawn. In the latter case, the user stake token has to be burnt, while in the former case, it has to be paid back to the **`UserStake`** script, together with the remaining stake. Since the script distinguishes the two cases by the presence or absence of a continuing output, and since the token name of the burnt token is not checked, it is possible to burn only one of the tokens when redeeming two UTxOs at once. The other token can then be transferred to an attacker.

Once a user token belongs to the attacker's public key, they can pay the token to the **`UserStake`** script with an arbitrary **`UserStakeDetail`** datum. An interesting part of the datum to forge is the vote map, and doing so makes it possible to vote with an arbitrarily high power without staking a corresponding amount of Gero assets, in positions that carry a genuine user token.

The token theft is illustrated in module **`Audit.AttackTraces`** in trace `stealTokenDuringDoubleClose`. An exploit that uses the stolen token to vote without staking the appropriate amount of Gero coins is shown in trace `freeVote`.

2.2.1.2 The minting policy of the `Gov` token mostly relies on trusting the administrator

Severity: Medium

The minting policy for the **`Gov`** NFT relies on the administrator to mint only one token and pay it to the **`Gov`** script. In principle, the administrator could mint several tokens and keep one (or more) to themselves. Even if a malicious administrator could only cause harm on the very first transaction, we think it advisable to enforce these conditions in the minting policy.

We classify this vulnerability as *medium* severity since it is possible and easy to inspect whether the administrator behaved as expected right after the first transaction is issued.

This concern is part of the audit test suite under the heading "attacks on the transaction that mints the Gov token".

2.2.2 Unclear Specification

2.2.2.1 User ids are not reused

Severity: Low

User identifiers (integers starting at 0, counted by `gsTotalStakers`), and therefore user token names, are not reused. This means that user token names can grow without bound, which does not scale well in the very long run.

2.2.2.2 No specification on burning `Gov` tokens

Severity: Low

The **`Gov`** tokens cannot be burnt at the end of the life cycle of the contract, and the specification does not state anything regarding closing **`Gov`** script outputs.

2.2.2.3 Unclear meaning of zero votes

Severity: Low

A user or delegatee may cast a vote with a power of 0 as shown in trace `voteZeroTr` in **`Audit.Traces`**. It is unclear from the specification whether a zero vote has a different intended meaning than casting a **`Nothing`**-vote (which removes a vote), and should consequently show up in the vote map together with the non-zero power votes.

2.2.2.4 `gsPolicyIds` is an unused and immutable piece of datum

Severity: Low

After the end of the audit, we were informed that the form of this datum is necessary for the off-chain part of the protocol, which was not part of the audited code.

The datum of **`Gov`** script outputs carries a list of allowed currency symbols for rewards, as stated by the specification. There is currently no way to modify this list (none of the existing redeemers allow it), yet it is part of the datum instead of the script parameters. Furthermore, it is never used and takes up space on **`Gov`** outputs. Using the **`RewardAsAdmin`** redeemer, it is possible to pay a user any kind of asset, provided the output value exceeds the input value; such transactions do not even involve spending the **`Gov`** output containing the `gsPolicyIds` list. We suggest to clarify its role in the specification, move it to the script parameters if it is not expected to change, or remove it altogether if it is truly unused.

2.2.2.5 Specification mentions unimplemented features

Severity: Lowest

Some of the features mentioned in the section "Basic assumptions on the system" of `gov-docs.md` are not implemented. We quote the relevant parts of the specification here.

- *"We can record immutably that a proposal has been submitted to the governance protocol, and that stakeholders can vote on it. The proposer can submit a transaction that records the proposal details on-chain. Voters can then reference the proposal when they vote."*

  There is no dedicated mechanism for proposals (e.g. a proposal script or a proposal redeemer for the **`Gov`** script).

- *"We can record immutably that the admin has injected revenue into the governance protocol. The admin can submit a transaction that either locks revenue funds under the governance protocol or transfers revenue funds directly to stakeholders."*

  This does not specify where the funds should be locked. If they are locked on the **`Gov`** script, they cannot be retrieved afterwards (lack of a dedicated redeemer).

- *"We can record immutably that the admin has allocated funding to proposals in the governance protocol, and how much was allocated to each proposal. The admin can submit a transaction that allocates specific funding to specific proposals."*

  There is no dedicated mechanism for proposals (e.g. a proposal script with a dedicated "fund" redeemer).

2.2.2.6 ■ Various minor concerns with the specification

Severity: Lowest

Ambiguous use of greater than: $\geq$ or $>$ The specification in `gov-docs.md` sometimes uses ambiguous "greater than" wording mixed with "strictly greater than" and "less than or equal". We suggest to rephrase using either "greater than or equal" or "strictly greater than" to avoid confusion.

Unclear negative specifications The specification mentions several features of the contract that are *not* part of the governance protocol (namely fair revenue distribution and funding allocation to proposals). It is unclear whether the "We can not ..." sections in `gov-docs.md` refer to technical impossibilities, design choices, limitations of the current implementation, or planned features.

2.2.3 Code Quality

2.2.3.1 ■ Error reporting is lacking

Severity: Medium

Error reporting is lacking, especially in the **`UserStake`** validator which makes uses of custom quasi-quoted code (for "and" laziness). It is also lacking when it comes to failure of partial functions such as when outputs are wrongly ordered.

2.2.3.2 Dead code

Severity: Low

Similarly to issue 2.2.2.4, the code base contains a significant amount of dead code. For example, the following list illustrates some unused functions:

- In **`Contracts.Util`**:
  - `sameTxOut`
  - `rawDatumOf`
  - `hasValidator`
  - `ownAdress`
  - `headFilterValidatorInput`
- In **`Contracts.RawContext`**:
  - `getInputOut`
  - `getOwnInput`
  - `getOwnOutput` (and therefore `getTxOutAddress` as well)
  - `getTxOutDatumHash`

This is a concern because it increases the maintenance cost and the chance of introducing bugs by using some function that was never tested because it was dead code.

2.2.3.3 Undocumented code section regarding closing positions

Severity: Low

Comments and documentation are lacking for the **`UserStake`** validator. In particular, the pattern-matching case for the **`Withdraw`** transaction that closes the user position is separated from the other possible redeemers, including the regular **`Withdraw`** case for when some stake remains on the user position. We suggest to document this function and make explicit how all these cases complement each other, especially since this is the code where the critical vulnerability 2.2.1.1 originates.

2.2.3.4 Various minor code quality concerns

Severity: Lowest

Comment for `fromSpendContext` is wrong In module **`RawContext`**, the function `fromSpendContext` also carries `fromMintContext`'s documentation comment by mistake. Furthermore, these comments simply paraphrase the type of the documented functions without any added information, which would be desirable since the functions are exactly the same apart from their types.

Naming of function `currentTime` The function `currentTime` in module **`Contracts.Util`** actually extracts the end of the validity range of a transaction. The name is misleading, especially because the current time is not available to validators.

Error reporting is phrased positively Debug messages are phrased in a way that describes successful/expected cases such as "x and y match" if x and y are expected to match. Such error messages are displayed when conditions are violated, therefore "x and y match" would be displayed when they actually mismatch.

2.2.4 Other Concerns

2.2.4.1 Unclear role of `Gov` validator and token

Severity: High

As it currently stands, it is unclear what is the role of the **`Gov`** validator. Based on our partial investigations of the code base, it seems like the **`Gov`** validator has two responsibilities:

- It counts the number of user stake positions opened so far, using that number to enforce uniqueness of user stake tokens and positions.
- It constrains the initial datum on the user stake output and makes sure it is paid to the correct **`UserStake`** script.

Uniqueness of user stake positions is already ensured by their representation as UTxOs, from the **`UserStake`** script, suggesting the first responsibility above is superfluous. The second constraint could be enforced by the minting policy of the user stake tokens directly. If the **`Gov`** validator is indeed only responsible for the two items above, it could be possible to remove the **`Gov`** validator and token altogether, leading to a substantial reduction of the amount of code.

2.3 Conclusion

This report outlines the 13 concerns that we have gathered while inspecting the design and code of Gero Governance, pertaining to the code contained in the files listed in Table 2.1. As stated in Chapter 1, Tweag does not recommend for nor against the use of any work referenced in this report. The code is clear and well written. Nevertheless, the existence of *critical* and *high* severity concerns is a warning sign.