Celo Contracts Audit Release 5

OPENZEPPELIN SECURITY | AUGUST 27, 2021

Security Audits

In another round of auditing, the cLabs team asked OpenZeppelin to review and audit recent changes to the core contracts of the Celo protocol.

Scope

This audit was a diff audit, meaning that the scope of the audit was the difference across files in certain pull requests. **The pull requests audited in this phase were #8129, #7837, and #8060. Note that the audit only covered production solidity files. A more detailed scope is below.**

Within the various pull requests, the following files were considered in-scope for this audit:

```
Within PR #7837
packages/protocol/contracts/common/UsingRegistryV2.sol

Within PR #8060
packages/protocol/contracts/governance/Validators.sol, lines 166 an

Within PR #8129:
packages/protocol/contracts/common/interfaces/IRegistry.sol
packages/protocol/contracts/liquidity/GrandaMento.sol
packages/protocol/contracts/stability/Reserve.sol
packages/protocol/contracts/stability/StableToken.sol
```

If a file is not listed above, it should be considered out of scope for this audit.

Overview of the changes

The bulk of the changes to these contracts introduce a new stability mechanism, the GrandaMento contract. The motivation behind GrandaMento is that it allows large purchases and sales of CELO or stablecoins to/from the Celo treasury at a constant price. Discussions which led to GrandaMento noted that price slippage was too high for swaps of roughly 50k USD worth of CELO/stablecoins.

GrandaMento was proposed in CIP-0038. It works by allowing any user to create a proposal and deposit CELO or stablecoins for trade with the Celo Reserve. Each proposal must then either be approved, or may be cancelled by the approver. After approval, the proposal cannot execute until the veto period has passed. During the veto period, Celo Governance can cancel the proposal. Once the veto period has passed, anyone may execute a proposal on behalf of the proposer, exchanging their locked funds and sending the proceeds to them. Safeguards exist to ensure that the price at which the trade executes is close to the current fair market rate for CELO.

In addition to the GrandaMento contract and it's supporting infrastructure, PR #7837 introduced a new version of the UsingRegistry contract, UsingRegistryV2, which has more registered contracts than its previous version. PR #8060 includes a simple bugfix to allow testing of slashing functionality using the mycelo tool.

Assumptions

We assume that all un-audited parts of the celo protocol work as documented. For GrandaMento, we assume that oracles will return correct and up-to-date prices.

Privileged Roles

There are two privileged roles introduced in the pull requests reviewed here. Both are explained here within the context of the GrandaMento contract.

- The "owner" role is the general Celo governance contract, which allows all CELO holders to vote. In GrandaMento, the owner role has the power to cancel approved proposals, set trade size limits and price spread, set the approver role, and set the veto period. The veto period is

GrandaMento contract and the Celo Reserve safe. In GrandaMento, their sole job is to approve, or not approve, exchange proposals. Proposals cannot be executed until they are explicitly approved.

Vulnerabilities

Below, we list all the vulnerabilities found in this audit.

Critical severity

None 🙂

High severity

[H01] Large stablecoin amounts may get stuck

Within the flow of executing or cancelling a proposal to sell a stablecoin for CELO, the `getSellTokenAndSellAmount` function is used to call the stable token's `unitsToValue` function to convert the underlying stored number of "units" of stable token to a "value" of the stable token. This mechanism exists so that stored funds in the GrandaMento contract experience inflation as they wait for their respective proposals to be executed.

However, when dealing with large amounts of stable token, which is what GrandaMento is intended for, the `unitsToValue` function may overflow. The calculation in `unitsToValue` utilizes the "inflation factor", which is currently `1e24` (see function `getInflationParameters` in the cUSD contract). It also casts the `units` parameter to `newFixed`, which multiplies it by `FIXED1_UINT`, where `FIXED1_UINT` is also `1e24`. Finally, the `divide` function also used in the calculation multiplies the already scaled value by an additional `1e24`. This final multiplication, within the `divide` function, may potentially cause the calculation to overflow and revert.

Consider the case where a user is attempting to execute a sale of 200 billion cUSD for CELO. When the `unitsToValue` function is hit, the `units` will be on the order of `200000000000*1e18`, or `2e29`. Then, casting it to `newFixed` will make the value

Note that this issue becomes more likely when considering the CXOF token, which also has 18 decimals but is worth roughly 1/500 of cUSD. Thus, 200 billion CXOF is roughly 400 million cUSD, a much more likely number to be exchanged. This issue may worsen as currencies worth substantially less than 1 USD are added. For example, the Vietnamese Dong is worth roughly 1/23000 USD, so roughly 5 million USD worth would trigger the overflow.

Note also that as the inflation factor increases over time, this issue becomes slightly more likely.

Finally, note that this issue does not exist for creating a proposal, as proposal creation utilizes the `valueToUnits` function. This is the crux of the issue, which is that funds will be locked inside the GrandaMento contract in the event a user attempts to exchange too much. This applies both when attempting to execute a proposal, and when attempting to cancel it.

Consider paying close attention to these limits when setting maximum exchange amounts, and warning users that if they attempt to exchange too much, their funds may be locked in the GrandaMento contract. We understand that inflation is currently not being applied to stablecoins, but in the event that it is in the future, any limits set here should be re-evaluated for long term health of GrandaMento.

Update: *Fixed in pull request #8500.*

Medium severity

None.

Low severity

[L01] Changing veto period can affect existing proposals

The GrandaMento.sol contract introduced a Veto Period as a time buffer between a proposal's approval and execution. This concept is critial from a proposer's point of view and should not be changed after a proposal is proposed. However, the contract owner can update this value at anytime by calling the function `setVetoPeriodSeconds` which will lead to a change of veto

proposals under a new veto period.

Considering baking the veto period into the proposal once it is created so it is not affected by future veto updates.

Update: *Fixed in pull request #8369.*

[L02] Exchanger cancellations can be front-run

When a proposal is created in GrandaMento.sol, it can only be cancelled before it is approved by the approver role by calling the `cancelExchangeProposal` function. However this process can be front-run by the approver role, who can approve the proposal and prevent the user from cancelling it.

Consider informing exchangers that this is possible. Consider giving proposals a grace period in which the exchangers can cancel them before the approver can approve them.

Update: *Acknowledged and fixed in issue #8328. The cLabs team has determined that informing exchangers of this possibility is sufficient.*

[L03] Proposal might suffer compounded loss when cancelled

When cancelling a proposal, the `cancelExchangeProposal` function calls `getSellTokenAndSellAmount` to return the target token and token amount for the refund. In L408 of `getSellTokenAndSellAmount`, a mechanism is implemented to cap the returned value to the balance of GrandaMento contract in case the conversion from units to value for stable token ends up bigger than the contract balance.

However if there are multiple proposals being cancelled, the design of this mechanism will compound the potential loss to the very last proposal. Although the value is unlikely to be significant, the design seems inconsiderate.

Consider changing the design to better handle potential conversion loss. For instance, consider using an accounting system which only depends on “units” rather than “value”, thus avoiding imprecision from conversions.

> whereas Granda Mento transfers are expected to be on the order of hundreds of thousands/millions cUSD.
> Due to StableToken's inflation mechanism, there's no way to transact directly in units, a(n inherently imprecise) conversion to value is necessary.

[L04] Approved proposals may be executed at any time

After a proposal reaches the `Approved` stage, it can be executed at any time after the "veto period" ends. However, note that proposals do not expire. Thus, a proposal can lay dormant after the veto period until it is profitable to perform it.

This issue is of low severity due to a few factors which hinder it's probability. The first is that a proposal may also be cancelled at any time after it is approved. Another factor is that anyone can execute a proposal once it is approved.

We mention this issue to make it clear to users that dormant proposals can lead to abuse. In the event that the network is experiencing high congestion, it may seem like a waste to approve or cancel a proposal due to high fees. However, it should be made clear that dormant proposals being executed at rates that differ substantially from current price of CELO can destabilize markets and drain the Celo reserves. Consider informing users, and perhaps establishing a proposal expiry time, after which approved proposals can no longer be executed.

Update: *Acknowledged, will not fix. The cLabs team's statement for this issue:*

> After brief discussion, we think this is worth informing users (both exchangers & community members) this can happen, but won't make any smart contract changes to implement an expiry (so essentially a wontfix). We believe that adding an expiry could leave the exchanger with more opportunity for malfeasance. If the price of the asset being bought has fallen drastically since the locked-in price, an exchanger could intentionally try their best to have the proposal expire so it no longer has a "losing" trade. Regardless, we expect usage of GrandaMento to be infrequent & high visibility, so it's likely that exchanges would be executed quickly when possible.

[L05] Be careful of noncompliant ERC20's

Although all current stable tokens, as well as the native asset CELO, do return `true` on a successful transfer, it is a common pattern within DeFi that some tokens do not return `true`. Thus, GrandaMento will not work for these tokens.

Additionally, any non-standard tokens which charge fees on transfer may not be accounted correctly, due to not checking the contract's balance post-transfer.

Consider checking for ERC20 compliance before allowing a token to be used with GrandaMento. Additionally, consider modifying the logic in GrandaMento to account for tokens which may charge fees by checking the change in the GrandaMento contract's balance before and after transfer.

Update: *Acknowledged. The cLabs team's statement for this issue:*

> It is presumed that the only ERC-20 tokens that will be allowed in Granda Mento are core Celo stable tokens that are based on StableToken.sol and approved by Governance. There's no general way to determine ERC-20 compliance on-chain, this needs to be done by humans off-chain and approved via a Governance vote.

[L06] No checks in `setMaxApprovalExchangeRateChange`

Within the function `setMaxApprovalExchangeRateChange`, there are no checks on acceptable values for `maxApprovalExchangeRateChange`.

In the proposal approval process, the `rateChange` variable represents a percentage difference between the proposed exchange rate and the current exchange rate. Then, the `rateChange` must be less than the `maxApprovalExchangeRateChange`. If `maxApprovalExchangeRateChange` is too low, it is highly likely that proposal approvals will revert.

Consider setting a lower bound for `maxApprovalExchangeRateChange` within `setMaxApprovalExchangeRateChange` to prevent excessive reversions during the proposal approval process.

Update: *Acknowledged, will not be fixed. The cLabs team's statement for this issue:*

> hardcoding a minimum doesn't help much. For example, a hardcoded minimum of 1% doesn't do a whole lot, and a higher value of 10% would paralyze the flexibility Governance has in restricting proposals easily.

[L07] No checks in `setVetoPeriodSeconds`

The `setVetoPeriodSeconds` function contains no checks on what value `vetoPeriodSeconds` can be set to.

The purpose of the veto period is to ensure community approval before executing a proposal. A proposal cannot be executed until the veto period has passed.

In the case that the veto period is `0` or too small, there will not be enough time for the community to veto malicious proposals before they are executed. In the event that the veto period is too high, proposals will be forced to wait potentially indefinitely long before being able to be excuted. This also has the effect of locking exchanger funds in the protocol, as exchangers are not able to cancel proposals once approved and thus release their funds.

Consider setting reasonable bounds for the veto period. The veto period's lower bound should be long enough that interested users are able to learn about and discuss the proposal, as well as get their transactions mined during periods of high network congestion. The veto period's upper bound should be short enough that an exchanger's funds cannot be locked in the protocol for too long, in the case that their proposal is approved.

Update: *Fixed in pull request #8368. A maximum bound has been added for the veto period.*

Notes & Additional Information

[N01] Overloaded error message

This error message is too vague. The error message, `"Sender cannot cancel the exchange proposal"`, does not provide enough information for a user to correct the problem.

This error subtly handles 3 separate failure cases:

- When the proposal is in the `Proposed` state and the `msg.sender` is not the exchanger.

When the proposal is in any state aside from `Executed` and `Failed`.

Error messages should be informative enough that a user understands and is able to correct the error. Consider splitting up this `require` into multiple cases with their own error messages, to help users troubleshoot in the event of a revert. This can be done with `if/else` clauses and `require` statements.

Update: *Fixed in pull request #8344.*

[N02] Unclear comments

Within the `GrandaMento` contract, this comment appears to be unfinished, and this comment could be clearer, specifying that the user who is performing the trade is the one "selling" (rather than the protocol being the one "selling").

Consider updating the two comments identified above to make the codebase clearer for reviewers and future developers.

Update: *Fixed in pull request #8344.*

Conclusions

No critical and two high severity issues were found. Some changes were proposed to modify the design of GrandaMento to reduce the potential attack surface. Some changes were proposed to help follow best practices, and some issues were reported to inform users of potentially unseen risks of the GrandaMento contract.

OpenZeppelin

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenZeppelin

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

OpenZeppelin

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits