Global Messaging Token Audit

OPENZEPPELIN SECURITY | OCTOBER 16, 2017

Security Audits

The Mercury Protocol team asked us to review and audit their Global Messaging Token (GMT) and crowdsale contracts. We looked at the code and now publish our results.

The audited contracts are located in the MercuryProtocol/global-messaging-token-contracts repository. The version used for this report is the commit `d0765cbd0732453832455dae0e2cf892da1ab572`.

Here's our assessment and recommendations, in order of importance.

Update: *The Dust team has followed most of our recommendations and updated the contracts. The updated version is at commit* `df08a450e3960d6717348d38da10b542d4123534`.

Critical Severity

Team can circumvent refund restriction

The `refund` function allows investors to ask for a refund if the minimum cap is not reached. Radical App International is given a share of tokens at the beginning of the process for which they should not be entitled to a refund. This is accounted for by not allowing their address to call `refund`. Since tokens are always transferable, they could easily circumvent this by transferring the tokens to another address and calling `refund` from it.

Consider disallowing transfers until the crowdsale ends successfully, for example by using OpenZeppelin's `PausableToken` or something similar. Not only will it fix this bug, but it's also

finalization of the crowdsale.

Update: *Fixed with the alternative suggestion in the latest version.*

High Severity

No issues of high severity.

Medium Severity

Arbitrary and redundant stage state variable

The functions `startSale`, `stopSale` and `setFailedState` allow the owner to set the `stage` state variable. However, there are no restrictions on when they can be called, and consequently the value of the `stage` variable isn't necessarily the *actual* stage of the sale. As an example, in the middle of the sale the owner can call `setFailedState`, without it having really failed.

The variable actually serves no purpose other than giving the owner some control over when its functions can be called. This control is limited, however, because there are already other restrictions in place. For example, `refund` can only be called after the sale period and if the minimum cap isn't reached. This makes us think the `stage` variable is redundant.

We would recommend to remove it along with the setter functions, to further trust minimization and remove the possible inconsistent states. Alternatively, add checks to `stage` setter functions, and only rely on those for other function preconditions.

Update: *The variable was removed in the latest version.*

Low Severity

Using block numbers to specify start and end

The sale uses block numbers to specify when it starts and when it ends. The current recommendation is to use timestamps instead. The risk of miner manipulation of timestamps is very low for this use case, and due to the Difficulty Bomb it is now very difficult to correctly estimate future block times. Consider switching to timestamps.

Consider performing sanity checks to validate `GMToken`'s constructor parameters. Check that `_startBlock < _endBlock`.

Update: *Fixed in the latest version.*

Reuse open source contracts

The contracts `StandardToken`, `GMToken` and `SafeMath` are very similar to code found in OpenZeppelin's `StandardToken`, `Crowdsale`, RefundableCrowdsale and `SafeMath` contracts. Reimplementing functionality instead of reusing public and already audited code can bring regression problems and difficult to find bugs. Consider installing and using the code available in OpenZeppelin.

Update: *The team has pointed out that the token is in fact taken from ConsenSys.*

Unsafe math

There are many unchecked arithmetic operations in `GMToken` and `StandardToken`. It's always better to be safe and perform checked operations. Consider using the `SafeMath` library, or performing pre-condition checks on any math operation.

Update: *Fixed in the latest version.*

ERC20 compliance

ERC20 specifies `decimals` to be a value of type `uint8`. It is declared as a `uint256` variable in `GMToken`. Consider changing it to `uint8`. If so, it will be necessary to cast to `uint256` when using the variable for arithmetic such as when expressing token amounts like in `500 * (10**6) * 10**decimals`. Consider defining a constant `uint256 TOKEN_UNIT = 10 ** uint256(decimals)` to write `500e6 * TOKEN_UNIT` in these cases.

A `Transfer` event is emitted next to `ClaimGMT` in the constructor, and there is a spot on comment with an explanation. We would add to it that emitting a `Transfer` event when creating tokens enhances user experience by allowing applications such as Etherscan to learn of the new

Update: *Fixed in the latest version.*

Duplicate decimals value

The `decimals` parameter used to calculate token amounts is duplicated in the `GMToken` and `GMTSafe` contracts. This redundancy is error-prone, as the two variables could get out of sync if they are changed at any moment. Consider leaving only the `decimals` variable defined in `GMToken`. The `GMTSafe` contract can store the token grain amount to be transferred, instead of the token unit amount (e.g. `4*10¹⁸ vs 4`).

Update: *Fixed in the latest version.*

Notes & Additional Information

- There are a couple of `TODO` notes in the contracts, with things to change before deployment. This is prone to forgetting, and we recommend to add constructor parameters for these pending values to avoid this kind of mistake.
- Some of the checks in `SafeMath` are redundant and can be removed: bothchecks in `div`, and the second check (`c >= b` in `add`.
- Keep in mind that there is a possible attack vector on the `approve` / `transferFrom` functionality of ERC20 tokens, described here. Consider implementing one of the proposed mitigations, or using the ERC20 implementation from OpenZeppelin which already has one in place.
- `transfer` checks that destination is not `0x0` or the token itself, but `transferFrom` doesn't. Consider adding the same checks there.
- In `unlock` in `GMTSafe` an ad-hoc "revert" was implemented, but the EVM's `revert` functionality could be used.
- In `GMTSafe`, the constructor parameter and `gmtAddress` state variable could be directly declared of type `GMToken`. If the constructor is declared as `GMTSafe(GMToken _gmt)` the ABI will remain `GMTSafe(address)`.
- `refund` has a boolean return value that is unused. Consider removing it.

Update: *Most suggestions were implemented in the latest version.*

Some changes were proposed to enhance standards compliance, follow best practices and reduce potential attack surface.

If you're interested in discussing smart contract security, join our slack channel, follow us on Medium, or apply to work with us! We're also available for smart contract security development and auditing work.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Global Messaging Token contracts. We have not reviewed the related Mercury Protocol or Dust projects. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Related Posts

Beefy Zap Audit

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

Security Audits

OpenBrush Contracts Library Security Review

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

Security Audits

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