SGame Token and Crowdsale Audit

by MonteLabs

October 21, 2018

1 Introduction

This document is a report representing the security audit conducted on the Game Loot Network crowdsale smart contracts, and discusses potential issues that we found while analysing and testing the code. Besides the basic token and crowdsale functionalities, the system involves complex logic regarding token bonuses, daily limits for the multisig wallet, and an upgradeable token.

The code was written by Pactum, and is hosted at `github.com/MDionSijan/GLN/`. The verified version is commit 5646dbc00a44c9604afcdf94edcc5f7f2c13dbe7 from March 22, 2018.

We divide the smart contracts into three components when discussing our findings: (i) GLNToken, (ii) Crowdsale and (iii) MultiSigWallet.

The GLNToken and Crowdsale contracts use OpenZeppelin smart contract libraries which are considered standard and have been heavily audited and used by the community in general. Figure 1 shows the inheritance relations between the used smart contracts, where the red nodes are external libraries and the green nodes are the newly implemented smart contracts, object of this audit.

The code has a high quality and an impressive test coverage, which is critical given the complexity of some of the features in these smart contracts. We have not found any vulnerabilities that would lead to loss of funds or control, and the high severity issues we list have the perspective of the user.

Section 2 lists the issues found during this audit for the three components, separated by severity level (High, Medium or Low) or as minor suggestions to improve code readability or remove redundant code Section 3 presents our concluding thoughts on the audit.

2 Issues

2.1 WhitelistedCrowdsale

2.1.1 Medium Severity

Crowdsale owner may mint extra tokens. The `GLNToken` contract derives from `MintableToken` but function `finishMinting()` is never called, which allows the owner of the contract to mint more tokens even after the crowdsale

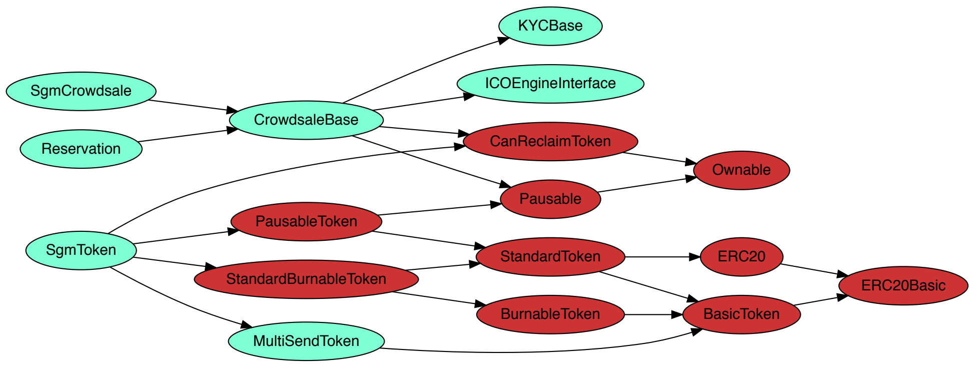

Figure 1: Inheritance relationship between the smart contracts used in the GLN ICO.

is ended. Our suggestion is to add the line `finishMinting();` inside function `endCrowdsale` in `WhitelistedCrowdsale.sol` so that the users have the contract guarantee that no more tokens will be minted.

2.1.2 Low Severity

Pre-sale may breach softcap and hardcap. In contract `WhitelistedCrowdsale`, function `preSaleUpdate` does not check the raised amount of `wei` against `softCap` nor `hardCap`. Even though unlikely, an unseen breach of soft cap or hard cap leads to features not working correctly.

Usage of `timestamp`. Even though it is known that variable `block.timestamp` may be manipulated by some minutes we do not consider it a problem in these contracts.

Every token purchase logs `CrowdsaleHasStarted`. In contract `WhitelistedCrowdsale`, function `buyTokens` always calls function `setState` which checks whether the crowdsale may be open. Regardless the state, function `setState` always emits event `CrowdsaleHasStarted`, which may be confusing or incorrect in the worst case.

2.1.3 Minor Suggestions

- In `WhitelistedCrowdsale.sol:262`, instead of reverting, the code could automatically let the beneficiary buy only the amount of tokens left to reach the hard cap and transfer the change back.

- In `WhitelistedCrowdsale.sol:303` the comment for function `addParticipant` says “This must be done for all participants before the crowdsale begins”, but the modifier `whenNotEnded` is used. The specification does not state which one is correct.
- In `WhitelistedCrowdsale::preSaleUpdate` no time constraint is used, so the pre sale still runs parallel to the crowdsale.
- In `WhitelistCrowdsale::editSoftSalesCap`, modifier `whenSoftCapNotBreached` can be removed, since `softCap` can only be breached during crowdsale and the following condition
  `require (crowdsaleState == CrowdsaleState.waitingToStart);`
  is even stricter.
- In `WhitelistCrowdsale::setTokenManager`, the used `assert` would be better as a `require`, since the function may be called at any point and the condition is used to filter undesired input, and not to represent a bug.
- In `WhitelistCrowdsale.sol:349` the whole struct may be deleted in order to save gas.
- Line `WhitelistCrowdsale.sol:469` may be removed, since it zeroes a local variable that is not used again.
- Function `WhitelistedCrowdsale::createTokenContract` is only used inside function `setTokenContract` and has just one line. These functions could be merged.
- Function `WhitelistCrowdsale::forwardFunds` has just one line that can be inlined where the function is called, since that is its only use.

2.2 GLNToken

2.2.1 High Severity

Contract `UpgradeAgent` does not enforce 1:1 token swap. The specification says that the users can upgrade their tokens from the `GLNToken` to another token that would be deployed in the future, an `UpgradeAgent`, in a 1:1 manner. The users have no guarantees of this feature whatsoever, and have to simply trust that the newly deployed token contract has a faithful implementation. A solution would be to have the crowdsale contract deploy the new token as well as the initial `GLNToken`, such that the logic behind the 1:1 token swap is guaranteed.

2.2.2 Minor Suggestions

- In `UpgradeableToken.sol:73,173` `require` should be used instead of `assert`, since the goal of the condition is to filter out unwanted input.

2.3 MultiSig Wallet

2.3.1 Low Severity

MultiSigWallet accepts address 0 as owner. In `MultiSigWallet.sol:108`, an address is accepted as an owner of the wallet if it's not 0 or if it is not an owner yet. If 0 is sent as an owner, it will be accepted the first time it is seen. The suggestion is to change the condition to `require (!isOwner[_owners[i]] && _owners[i] != 0);`.

MultiSigDailyLimit should use `SafeMath`. Contract `MultiSigDailyLimit` should use `SafeMath` for a clearer overflow handling in lines 73, 80, 100 and 119.

2.3.2 Minor Suggestions

- In `MultiSigWallet.sol:94` condition `_ownerCount != 0` is redundant, since `_required <= _ownerCount && _required != 0` are part of the constraints.
- Function `MultiSigWallet::isConfirmed` should return `false` after the `for` loop.
- The `if` in `MultiSigDailyLimit.sol:117` could be removed since daily limit will never be greater than `spentToday`, but equal at maximum.

3 Conclusion

We have analysed the Game Loot Network crowdsale smart contracts involving three different components: the token, the crowdsale, and a multisig wallet. The code is well written and easy to read even though some of the features are rather complex. We have found no vulnerabilities that might lead to loss of funds or control over the contracts. The two main issues we found were categorized as high/medium severity from the perspective of the user, since they would have to trust certain behaviors that are currently not enforced by the smart contracts. The first main issue can be quickly fixed, and the second requires a conceptual decision on how it has to be. The minor issues we list are easily fixable, if the developers agree with our claims.

This document is stored on IPFS as a security evidence. This audit also implies that MonteLabs issues an on-chain audit verification seal that can be accessed directly via the smart contract or at montelabs.com/audits as soon as the GLN contracts are deployed, where all the security evidences can be fetched.