INFINIGOLD PTY LTD

Smart Contract Changes

Security Review

Version: 1.0

November, 2020

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of an update made to the smart contracts underlying the Infinigold platform, a gold-backed stablecoin powered by Ethereum.

Sigma Prime previously performed two time-boxed security reviews of the smart contracts underlying the Infinigold token (PMGT).

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Overview

InfiniGold allows investors to buy, sell and hold physical gold stored at The Perth Mint - a large refining mint - in a digital form. To support this gold-backed stable coin, Inifingold has developed a set of smart contracts on Ethereum which implement the following:

- The ERC20 standard [1];
- Minting functionality, restricted to authorised addresses (i.e. minters);
- Burning functionality, invoked when an ERC20 transfer is made to a specific burn address;
- Proxy pattern to support contract upgrades based on *ZeppelinOS* [2];
- Separate storage contracts for token balance sheet, token allowances and parts of role-based access control;
- Blacklisting feature allowing accounts assigned to the `ListerRole` role to prevent specific addresses from interacting with the smart contracts. *Update: the last changes introduced the ability for a contract owner to transfer funds from blacklisted addresses.*;
- Whitelisting feature enabling accounts assigned to the `ListerRole` role to allow specific addresses to burn tokens (by transferring to the `burnAddress`).

The related Perth Mint Gold (PMGT) tokens is redeemable for gold certificates, with operational procedures for exchanging between them. It is understood that the Infinigold stablecoin is *centralised*, since smart contracts include operational roles for facilitating actions such as token minting.

Security Assessment Summary

The scope of this review is solely focused on the changes introduced by the Infinigold development team to the PMGT smart contracts to provide the ability for the contract owner to transfer funds from blacklisted addresses.

The SHA256 hash of the archive containing the updated contracts is:

`50d90fb5bd981d7f512d4c859a9a9bff4c6613a900ee0274e5619a4ffe9fce98`

The changes are limited to the addition of the following functions:

- **TokenImpl.sol**: `transferFromBlacklisted()`
- **Blacklistable.sol**: `_transferFromBlacklisted()`

The manual code-review section of the report focused on identifying any issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focuses on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [3, 4].

During this review, the testing team identified one (1) informational issue, acknowledged by the development team.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the scope of this review. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
IBL-01	Unnecessary `require` Statement	Informational	**Closed**

IBL-01	Unnecessary `require` Statement
Asset	Latest update `Blacklistable.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The following `require` statement in `Blacklistable.sol` can be removed:

- line [106]: `require(to != address(0));`

Indeed, this check is implemented in the `_transfer()` function in the `ERC20.sol` smart contract.

Recommendations

Consider removing the require statement on line [106] for gas saving purposes.

Resolution

The development team provided the following response:

"The development team acknowledges this gas inefficiency but notes that the Blacklistable.sol has been designed as a reusable contract that may not always be used with the current ERC20.sol implementation, and as such the development team has made the decision to keep the redundant statement to ensure any future usages of Blacklistable.sol are secured appropriately. The team also notes that the new functionality will only be used by an InfiniGold operational account in very rare circumstances, hence the additional gas cost is negligible"

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] ERC-20 Token Standard. Github, Available: `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md`.

[2] Open Zeppelin. Proxy Patterns, Available: `https://blog.zeppelinos.org/proxy-patterns/`. [Accessed 2019].

[3] Sigma Prime. Solidity Security. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html`. [Accessed 2018].

[4] NCC Group. DASP - Top 10. Website, 2018, Available: `http://www.dasp.co/`. [Accessed 2018].

[5] Sigma Prime. Solidity Security - Delegatecall. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html#delegatecall`. [Accessed 2018].

[6] Sigma Prime. Solidity Security - Front Running. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html#race-conditions`. [Accessed 2018].

[7] NCC Group. DASP - Front Running. Website, 2018, Available: `http://www.dasp.co/#item-7`. [Accessed 2018].

[8] OpenZeppelin StandardToken.sol. Github, 2018, Available: `https://github.com/OpenZeppelin/openzeppelin-solidity/blob/v2.1.2/contracts/token/ERC20/ERC20.sol`.

σ'