Saga's Smart Contracts Security Audit

Prepared by

Alex Bashlykov
CTO, Zerion

October 15th, 2019

Abstract

This report presents the results of the security audit of the **Saga's Smart Contracts** conducted by the **Zerion** team.

The whitepaper of the project is available at https://www.saga.org/static/files/saga-whitepaper.pdf.

The description of the monetary model can be found at https://www.saga.org/static/files/saga-monetary-model.pdf.

The analyzed source code is available at a private GitHub repository: https://github.com/SagaCore/zerion-audit-saga-contracts-1.

The version used for this report is commit:

#c9214728f7ef66ed12bdbf476197782180dccd4d (Aug 15, 2019)

Methodology

1. We use automated tools such as SmartCheck and Remix for static code analysis.
2. We manually inspect smart-contracts for known security vulnerabilities.
3. We analyze the provided whitepaper and other documentation and compare the implemented logic with the one described.
4. We find all possible entry points, both public and permissioned, and manually verify that every flow works as expected.
5. We reflect all the findings in this report.

Disclaimer

The audit does not give any warranties on the security of the code. Considering the size of the project, findings brought up to the team are not to be considered exhaustive. We always recommend proceeding with multiple independent audits and a public bug bounty program to ensure the security of the smart contracts.

Project Structure

- authorization
  - AuthorizationActionRoles.sol
  - AuthorizationDataSource.sol
- contract_address_locator
  - ContractAddressLocator.sol
  - ContractAddressLocatorHolder.sol
  - ContractAddressLocatorProxy.sol
- saga
  - ETHConverter.sol
  - IntervalIterator.sol
  - MintingPointTimersManager.sol
  - MintManager.sol
  - ModelCalculator.sol
  - ModelDataSource.sol
  - MonetaryModel.sol
  - MonetaryModelState.sol
  - PaymentManager.sol
  - PaymentQueue.sol
  - PriceBandCalculator.sol
  - RateApprover.sol
  - ReconciliationAdjuster.sol
  - RedButton.sol
  - ReserveManager.sol
  - SGAAuthorizationManager.sol
  - SGAToken.sol
  - SGATokenManager.sol
  - SGAWalletsTradingLimiter.sol
  - TransactionLimiter.sol
  - TransactionManager.sol
- saga-genesis
  - SGNAuthorizationManager.sol
  - SGNConversionManager.sol
  - SGNToken.sol
  - SGNTokenManager.sol
  - SGNWalletsTradingLimiter.sol
- utils
  - Adminable.sol
- wallet_trading_limiter
  - TradingClasses.sol
  - WalletsTradingDataSource.sol
  - WalletsTradingLimiterBase.sol
  - WalletsTradingLimiterValueConverter.sol

There are several external dependencies that were not audited, such as **Claimable**, **ERC20**, and **SafeMath** by OpenZeppelin, as well as a **MultiSigWallet** by ConsenSys.

Public Entry Points

There are multiple public entry points that can be called externally to start interacting with the system.

SGNToken

- transferFrom()
- transfer()
- convert()
- approve() [ERC20]
- increaseAllowance() [ERC20]
- decreaseAllowance() [ERC20]

SGAToken

- withdraw()
- deposit()
- transferFrom()
- transfer()
- exchange()
- ()
- approve() [ERC20]
- increaseAllowance() [ERC20]
- decreaseAllowance() [ERC20]

Permissioned Entry Points

These are the entry points that have restricted access and can only be called from specific addresses.

authorization.AuthorizationDataSource

- removeAll(), only admin
- upsertAll(), only admin
- removeOne(), only admin
- upsertOne(), only admin
- **Adminable** + **Claimable** entry points

contract_address_locator.ContractAddressLocatorProxy
- upgrade(), only owner
- **Claimable** entry points

saga.ETHConverter
- setPrice(), only admin
- **Adminable** + **Claimable** entry points

saga.MintManager
- updateMintingState(), only authorized for public operations addresses

saga.ModelDataSource
- setInterval(), only owner
- lock(), only owner
- **Claimable** entry points

saga.PaymentManager
- settlePayments(), only authorized for public operations addresses
- setMaxNumOfPaymentsLimit(), only owner
- **Claimable** entry points

saga.PaymentQueue
- clean(), only authorized for public operations addresses

saga.RateApprover
- setRateBounds(), only owner
- **Claimable** entry points

saga.ReconciliationAdjuster
- setFactor(), only owner
- **Claimable** entry points

saga.RedButton
- setEnabled(), only owner
- **Claimable** entry points

saga.ReserveManager
- setThresholds(), only owner
- setWallets(), only owner
- **Claimable** entry points

saga.SGAWalletsTradingLimiter
- **Claimable** entry points

saga.TransactionLimiter
- setMaxDiff(), only owner
- **Claimable** entry points

saga-genesis.SGNWalletsTradingLimiter
- setSGNMinimumLimiterValue(), only owner
- **Claimable** entry points

wallet_trading_limiter.TradingClasses
- setLimit(), only owner
- **Claimable** entry points

wallet_trading_limiter.WalletsTradingDataSource
- resetWallets(), only admin
- **Adminable** + **Claimable** entry points

wallet_trading_limiter.WalletsTradingLimiterValueConverter
- setPrice(), only admin
- **Adminable** + **Claimable** entry points

Results

We iteratively worked with the Saga team reviewing their changes and providing the feedback. All problems and inconsistencies discovered by our team in the course of the work were resolved by the developers. We want to highlight the high readability of the code as well as the great quality of the provided documentation. The source code was significantly improved since the first revision given to us and we assert that we found **no critical issues** in the final version of the analyzed smart-contracts.