Dala Crowdsale Smart Contract Audit

1. Introduction

iosiro was commissioned by Wala to conduct an audit on their token and crowdsale smart contracts for their Dala token ICO. The audit was performed between 05 October 2017 and 13 October 2017.

The Dala token allows free banking and remittances for emerging market consumers. It is a general-purpose, open-source ERC-20 crypto-token. More information on the Dala token can be found in the whitepaper here.

This report is organized into the following sections.

- **Section 2 - Executive Summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit Details:** A description of the scope and methodology of the audit.

- **Section 4 - Design Specification:** An outline of the intended functionality of the smart contracts.
- **Section 5 - Differential Analysis:** An overview of notable differences found during the differential analysis.
- **Section 6 - Detailed Findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the risk exposure of the smart contracts, and as a guide to improve the security posture of the smart contracts by remediating the issues that were identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

---

2. Executive Summary

This report presents the findings of an audit performed by iosiro on the Dala token and crowdsale smart contracts. The purpose of the audit was to achieve the following.

- Ensure that the smart contracts functioned as intended.
- Identify potential security flaws.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. There are a number of techniques that can help to achieve this, some of which are described below.

- Security should be integrated into the development lifecycle.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed wherever possible.

The smart contracts used in the Dala token sale implementation were largely developed using previously audited projects, including projects by TokenMarket and OpenZeppelin.

At the client's request, a differential analysis was used to reduce the time required to conduct the audit. This approach comes at the expense of also reducing the scope of the audit to focus on previously unaudited code. A differential analysis allows one to identify code that has been introduced from a prior version of audited code. Testing of unaltered code was limited to a high-level functional verification, where the code was verified to operate as intended. Code that was altered for the specific implementation of the Dala token sale was fully audited.

The audit identified a number of points of interest. The code was well commented, modularized logically, and did not contain unnecessarily complex functions. At a high-level, both of the smart contracts operated as intended. No high or medium risk findings were identified during the audit. The low risk finding titled Pausing Functionality has a Single Point of Failure, could potentially have a high impact on the token. However, due to the difficulty of exploitation and trade off of the complexity of the proposed remedial action, the finding was limited to low risk.

Unfortunately, at the time of testing the production Truffle framework migrations were not available. As such, specific production parameters could not be recorded, such as the ether cap or the price of the tokens.

The risk posed by the smart contracts can be further mitigated by using the following controls prior to releasing the contracts to a production environment.

- Use a public bug bounty program to identify security vulnerabilities.
- Perform additional audits using different teams.

3. Audit Details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from any other files are considered to be out-of-scope.

3.1.1 Dala Corporation

Project Name: dala-smart-contracts

Commit: 175138b

Files: AllocatedCrowdsale.sol, CentrallyIssuedToken.sol, Crowdsale.sol, DefaultFinalizeAgent.sol, FinalizeAgent.sol, FlatPricing.sol, FractionalERC20.sol, Haltable.sol, MultiSigWallet.sol, MultiSigWalletWithDailyLimit.sol, NullFinalizeAgent.sol, PausableToken.sol, PricingStrategy.sol, ReleasableToken.sol, StandardToken.sol, UpgradeAgent.sol, UpgradeableToken.sol

3.2 Original Codebases

The following codebases were used to construct the audited source code. The original codebases were used in the differential analysis to detect changes.

TokenMarket

Project Name: ico

Commit: fa7feed

Files: AllocatedCrowdsale.sol, CentrallyIssuedToken.sol, Crowdsale.sol, DefaultFinalizeAgent.sol, FinalizeAgent.sol, FlatPricing.sol, FractionalERC20.sol, Haltable.sol, NullFinalizeAgent.sol, PricingStrategy.sol, ReleasableToken.sol, StandardToken.sol, UpgradeAgent.sol, UpgradeableToken.sol

Gnosis

Project Name: MultiSigWallet

Commit: b7f01af

Files: MultiSigWallet.sol, MultiSigWalletWithDailyLimit.sol

OpenZeppelin

Project Name: zeppelin-solidity

Commit: 6331dd1

Files: PausableToken.sol

3.3 Methodology

A variety of techniques were used to perform the audit, these are outlined below.

3.3.1 Dynamic Analysis

The contracts were compiled, deployed, and tested using both Truffle tests and manually on a local test network. A number of pre-existing tests were included in the project. The results of the tests and the coverage can be found in Appendix II.

3.3.2 Automated Analysis

Tools were used to automatically detect the presence of potential vulnerabilities, such as reentrancy, timestamp dependency bugs, transaction-ordering dependency bugs, and so on. Static analysis was conducted using Mythril and Oyente. Additional tools, such as the Remix IDE, compilation output and linters were used to identify potential security flaws.

3.3.3 Code Review

Source code was manually reviewed to identify potential security flaws. This type of analysis is useful for detecting business logic flaws and edge-cases that may not be detected through dynamic or static analysis.

3.3.4 Differential Analysis

Code that was unchanged from the previously audited code was determined to be out-of-scope for the security audit. It should be noted that this process does not inherently verify whether these contracts are secure. The purpose of a differential analysis is simply to check whether potentially vulnerable code was introduced into the contracts.

3.4 Risk Ratings

Each Issue identified during the audit is assigned a risk rating. The rating is dependent on the criteria outlined below..

- **High Risk** - The issue could result in a loss of funds for the contract owner or users.
- **Medium Risk** - The issue results in the code specification operating incorrectly.

- **Low Risk** - A best practice or design issue that could affect the security standard of the contract.
- **Informational** - The issue addresses a lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been addressed to a satisfactory level to remove the risk that it posed.

---

4. Design Specification

The following section outlines the intended functionality of the smart contracts.

4.1 Dala Token

The Dala token is described below.

ERC20 Token

The token implements the ERC20 standard.

Field	Value
Symbol	DALA
Name	Dala
Decimals	18
Total Supply	1,000,000,000

Releasable

Used to enforce a lockup period of the tokens until the crowdsale finishes. Releasable tokens can only be transferred after a release command is sent to the contract. The command needs to be issued by the release agent that is set by the owner of the

contract. Addresses can be whitelisted to perform transactions prior to becoming releasable by setting the address as transfer agents.

Upgradable

Tokens can be upgraded through an opt-in process. Users would need to send tokens to the upgrade function, which would then issue the appropriate number of tokens on the newly issued token. This function provides the ability to upgrade the contract due to added functionality or security improvements.

Pausable

The tokens can be paused. This functionality allows the owner of the contract to prevent the transfer of all Dala tokens. Pause functionality can be useful if the contract becomes compromised in any way. In the event of a compromise, the owner can enforce a pause of the token, at which time users could upgrade their tokens to a newly patched contract.

4.2 Dala Crowdsale

The Dala crowdsale is described below.

Cap

The crowdsale was limited to a specific number of Dala tokens and a price was given to each Dala token. This resulted in an effective ether hard cap for the crowdsale. If this cap was reached, further attempts to send ether to the contract would fail and the crowdsale could then be finalized. The ether cap was not available at the time of the audit.

Whitelist

Participants had to manually be whitelisted by the owner of the contract before they could participate in the crowdsale. This was implemented as the organizers required participants to submit KYC data to a registration website to be manually vetted and approved, prior to participating in the event.

The registration website can be found here.

Daily Ether Cap

The crowdsale enforced a daily ether cap per participant, which would prohibit participants from purchasing Dala tokens worth more than a specified number of ether per day. The cap was used to allow a fair amount of time for all participants to purchase Dala tokens. The algorithm for calculating the cap is given below.

```
dailyEtherCap = baseEtherCap * ((2 ^ (daysSinceStart + 1)) - 1)
```

For example, with a `baseEtherCap` of 15 ether, the `dailyEtherCap` would be 15 on the first day, 45 on the second day, 105 on the third day and so forth.

The daily cap was not available at the time of the audit.

Flat Pricing

The crowdsale made use of a flat pricing strategy. Flat pricing results in all participants paying the same price for tokens throughout the duration of the crowdsale.

Finalizable

The crowdsale could manually be finalized by the owner of the contract after the cap was reached or the crowdsale end date passed. After the crowdsale was finalized the tokens became transferable.

Haltable

The contract inherited from TokenMarket's Haltable contract. This would allow the owner to halt the crowdsale in the event of an emergency, which would prevent both the issuance of new tokens and the ability to finalize the crowdsale.

4.3 Wallet

Multisig Wallet

In order to store the funds of the crowdsale safely, the crowdsale made use of a Gnosis multi-signature wallet to deposit the funds. A 2-of-3 signatures configuration with hardware wallets securing the private keys of the signing accounts was used.

5. Differential Analysis

The source code that was used to develop these smart contracts was largely used from codebases published by TokenMarket and OpenZeppelin. The code has been tested, audited and used by a number of projects. The TokenMarket ICO code has supported Civic, Storj , and Monaco. The OpenZeppelin code has supported Golem, Firstblood, and Signatura.

5.1 Notable Differences

A description of notable differences between the audited codebase and the original codebases is given below.

5.1.1 Crowdsale.sol

Added `baseEthCap` Functionality

An ether cap was implemented to restrict the number of tokens that could be purchased each day. The functionality operated as intended and no vulnerabilities were identified.

The functionality made use of block timestamps, which can be slightly altered by miners. The potential for exploitation seemed inconsequential, as it would only allow some miners to purchase their daily cap seconds before everyone else.

Added Whitelist Functionality

Functionality was added to limit participants to people who had previously registered and been accepted through a registration portal. This functionality operated as intended and no vulnerabilities were identified.

Fallback Function Changed from throw to `buy()`

The fallback function of Crowdsale.sol was changed from simply performing a throw to calling the `buy()` function. This would allow participants to send ether directly to the contract address to purchase their tokens. The functionality operated as intended and no vulnerabilities were identified.

In the unlikely event that the function was ever intended to be called directly from send() or transfer() the gas stipend of 2300 would be insufficient to purchase the token.

5.1.2 CentrallyIssuedToken.sol

Changed Token Inheritance

The inheritance of the token was changed to remove the burnable feature, and added the ability to release and pause it. The added features operated as intended and no vulnerabilities were identified.

Changing the files that are inherited or the order of inheritance can have unexpected effects. The Solidity compiler uses C3 linearization to force a specific order in the DAG of base classes. As such, without a good understanding of the functionality being used, it may be possible to introduce bugs through superclass calls. CentrallyIssuedToken was not found to be vulnerable to this style of attack. An example of this type of attack can be found here.

5.1.3 PausableToken.sol

No notable differences were noted on PausableToken.sol.

However, it was found that an outdated version was being used at the time of testing. The newer version had seen minor patches, including setting `uint256` in place of `uint`, and explicitly setting visibility on functions. These changes were done to follow best practice.

5.1.4 OpenZeppelin SafeMath

The original codebase used used SafeMath in a contract form, whereas the altered code used the library form. In newer versions of the original codebase, the intention appears to be to migrate across to the library form as well. There are some differences in the way that functions are called, but otherwise no noteworthy differences were found.

---

6. Detailed Findings

The following section includes in depth descriptions of the findings of the audit.

6.1 High Risk

No high risk issues were present at the conclusion of the audit.

6.2 Medium Risk

No medium risk issues were present at the conclusion of the audit.

6.3 Low Risk

6.3.1 Pausing Functionality Has a Single Point of Failure

PausableToken.sol and Haltable.sol

Description

The pausing functionality that is available to the crowdsale and the Dala token smart contracts are both invoked through a single owner of the contract. As such, there is a single point of failure if the owner account is either compromised or is lost due to unforeseen circumstances. This could result in a situation where a contract is effectively paused indefinitely. The risk of indefinite pausing is mitigated with the Dala tokens, as an indefinite pause could theoretically be circumvented by a contract upgrade issued through an upgrade agent.

Remedial Action

It is recommended that the functionality follows a multisignature behavior, where for example 2-of-3 accounts are used to pause or halt a contract. The additional code complexity would likely increase the attack surface, which presents a trade-off that needs to be considered.

6.4 Informational

6.4.1 Design Comments

General

The following describes possible actions to improve the functionality and readability of the codebase.

Low coverage of tests

Tests can provide some form of assurance that the code is performing as expected. In this instance, the tests were Truffle framework tests that would deploy and execute the code locally via testrpc.

The coverage was low for a number of the files, as ideally one would have complete coverage across all the files used. While testing should not be used as the only measure of functionality, it can be helpful in identifying faults in logic, both at a data flow and business level.

Multisignature wallet expands attack surface

MultiSigWallet.sol and MultiSigWalletWithDailyLimit.sol

The Gnosis multisignature wallet was used to store the funds received during the crowdsale. The wallet was implemented through the use of a smart contract. If any vulnerabilities are found within the implementation of the wallet, it could lead to a compromise of the funds raised during the crowdsale. An example of this threat was when the Parity wallet was hacked through the multisignature functionality that led to the loss of approximately $30,000,000 USD.

The security advantages of using a multisignature wallet seem to outweigh the increased attack surface, thus this finding is simply listed for completeness.

Token upgrade can be required through pause

PausableToken.sol

While the token upgrade functionality is theoretically an opt-in process for users, the owner of the token contract could require users to upgrade by placing an indefinite pause on the original token. This would prevent users from being able to transfer their original tokens. The only way they would be able to access their original tokens would be by upgrading their original tokens to the new token.

Unmarked visibility on functions and state variables

General

There were a number of instances where functions and state variables had no visibility set. An example of both of these issues can be found in PausableToken.sol . Best practice specifies that visibility should be set explicitly. The intention of this is to avoid confusion and potential mistakes that could occur as a result of incorrect usage.

Inexact solidity compiler version used

General

The pragma version was not fixed to a specific version, as it specified $\hat{}0.4.15$, which would result in using the highest non-breaking version (highest version below $0.5.0$). According to best practice, where possible, all contracts should use the same compiler version, which should be fixed to a specific version. This helps to ensure that contracts do not accidentally get deployed using an alternative compiler, which may pose the risk of unidentified bugs. An explicit version also helps with code reuse, as users would be able to see the author's intended compiler version. It is recommended that the pragma version is changed to a fixed value, for example $0.4.15$.

Using undocumented solidity behavior

SafeMath.sol

SafeMath currently relies on the undocumented behavior of overflows in Solidity. There is potential for this functionality to change in future revisions, which could render the result of SafeMath incorrect, or break entirely. The likelihood of this seems low, and in the event that it did happen, it would only affect newly compiled code. Compilers would also likely warn of this issue, which further mitigates the risk.

6.5 Closed

No closed issues were present at the conclusion of the audit.

Appendix I: Max Ether Cap Functionality Added

---

03 November 2017

Commit hash: f9fbafc

Description

Functionality was added to the crowdsale smart contracts to allow an absolute cap to be set on the amount of ether that an address could purchase during the crowdsale.

The audit revealed that all of the added functionality operated as intended. The cap would limit the amount of ether that participants could send to the crowdsale. The limit could be changed through the setter function, which would then enforce the new limit.

No security issues were identified in the added code.

Changes

AllocatedCrowdsale.sol

- The constructor now has a parameter for the maximum participation amount cap per address.

Crowdsale.sol

- Added `setMaxEthPerAddress(uint _maxEthPerAddress)`, a setter function that could be used to change the maximum participation cap.
- Added an event, `MaxEthPerAddressChanged(uint newMaxEthPerAddress)`, to indicate that the max ether cap had changed.
- The `getCurrentEthCap()` function was changed to take the max ether cap into account when calculating the dynamic cap.

Request a service

START NOW →

ABOUT

CONTACT US

SMART CONTRACT AUDITING

PENETRATION TESTING

AUDIT REPORTS

PRIVACY POLICY

TERMS OF SERVICE