Blockport Crowdsale Audit

January, 2018

Alexander Wade

Contents

1. Overview

This document serves as the official audit report of a crowdsale and token contract, created and published by Blockport. Blockport intends to create a new type of cryptocurrency exchange. This new model is hybrid-decentralized, meaning that Blockport intends to securely store users' funds using a decentralized exchange architecture, while utilizing external, centralized exchanges to provide liquidity. Additionally, Blockport's model includes a social component, where experienced traders are able to leverage their experience to other traders in exchange for Blockport Token (BPT).

The project audited is Blockport's crowdsale contract and token contract. Both contracts are heavily based in OpenZeppelin's existing templates and frameworks. The crowdsale contract defines a capped-finalizable model, along with whitelisted investing. This means that the crowdsale defines a maximum investment cap in wei, both per-investor and per-contract. Additionally, the crowdsale is locked to non-whitelisted participants, and tokens are locked until the crowdsale is complete.

Upon completion of the crowdsale, the Blockport team and Blockport company are each minted 20% of the total number of tokens minted during the sale.

2. Introduction

2.1 Authenticity

The audited contracts are in the Blockport tokensale repository: https://github.com/Blockport/tokensale. The version used for this audit is commit ab6f75df78735f8252d15068619c0797099e433b. The main contract for the crowdsale itself is located in BlockportCrowdsale.sol. The token contract is located in BlockportToken.sol. Both contracts inherit heavily from OpenZeppelin contracts, which are also located in the linked repository.

2.2 Scope

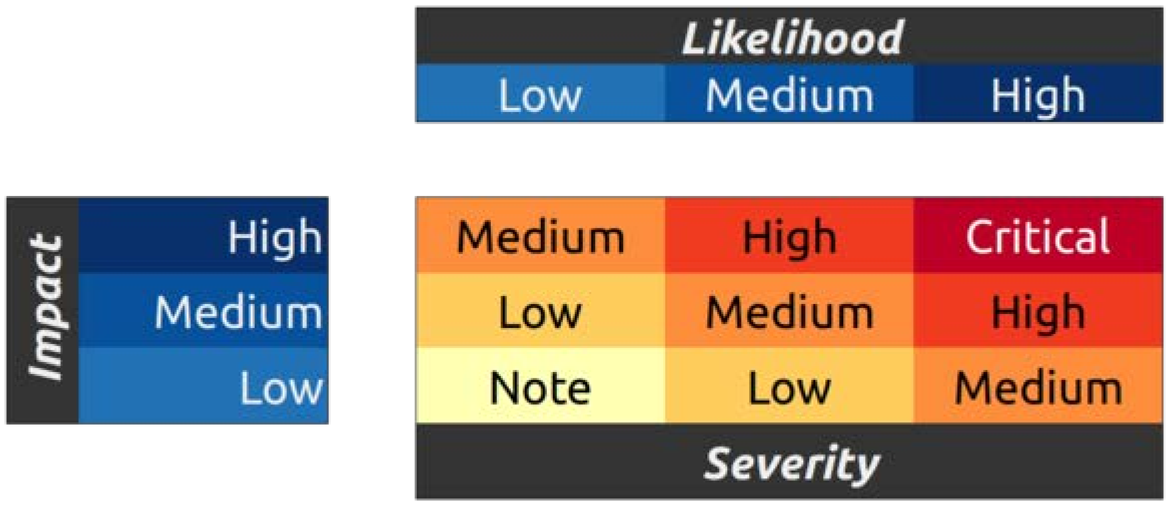
This audit covered most of the solidity files located in the contracts folder of the tokensale repository: https://github.com/Blockport/tokensale/tree/master/contracts. As the Blockport pre-sale had already taken place, the files BlockportPresale.sol and Whitelist.sol as well as their unique dependencies were omitted from this audit. Additionally, only the files in the zeppelin folder from which Blockport contracts inherited were included. A complete list of audited contracts can be found in the line-by-line commented code, located in the EthereumAuthio github repository, for which there is a link in the Documents and Resources section of this document.

2.3 Methodology

This audit focuses heavily on not only inspecting the smart contracts for vulnerabilities and potential for losses in funds, but also on working closely with the Blockport team to scrutinize the contracts for execution of intent. The end goal of this audit is to help the team not only secure their contracts, but also to ensure their vision for the project is best represented by the contracts they put forward. As a result, additional concerns such as efficiency and code design are included in this report as well.

2.4 Terminology

This audit categorizes vulnerabilities using the OWASP risk rating method based on impact and likelihood. Each vulnerability is given impact and vulnerability scores, which are used to give a more accurate estimation of a vulnerability’s overall severity. An additional factor in severity is the relative ease with which a vulnerability is fixed: an issue which requires extreme refactoring will be weighted higher than one with the same severity which is a quick fix.



	Likelihood		
Impact	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Note	Low	Medium
	Severity		

2.5 Disclaimer

This document reflects the understanding of security flaws and vulnerabilities as they are known to the Authio team, and as they relate to the reviewed project. This document makes no statements on the viability of the project, or the safety of its contracts. This audit is not intended to represent investment advice.

3. Findings

3.1 General

The Blockport contracts were impeccably put together, well organized, and heavily commented. The quality of both the code and attention to detail are very high. It was simple to read through the provided contracts and understand both function and intent.

3.2 Contract Explanation

3.2.1 Function - Crowdsale

BlockportCrowdsale.sol:
This contract houses the majority of the logic for the crowdsale itself. Users purchase functions by calling the fallback function, which is defined in the Zeppelin contract `Crowdsale.sol`. `BlockportCrowdsale` inherits from OpenZeppelin's `CappedCrowdsale.sol` to give it a maximum cap of wei to raise, as well as OpenZeppelin's `FinalizableCrowdsale.sol` to give the crowdsale defined logic post-finalization. Both of these contracts are located in the tokensale zeppelin folder.

3.2.2 Function - Whitelist

CrowdsaleWhitelist.sol:
This contract dictates the logic surrounding updating the crowdsale whitelist. Crowdsale participants are checked against a whitelist mapping during every purchase, which is updated in this contract using `addToWhitelist` and `removeFromWhitelist` functions.

3.2.3 Function - Token

BlockportToken.sol:
This contract defines the Blockport Token (BPT) itself. It inherits primarily from

OpenZeppelin's `CappedToken` and `PausableToken` contracts, and does not define any additional methods aside from the constructor.

3.3 Critical Severity

No critical severity issues were found.

3.4 High Severity

***UPDATE JAN 25:** On Jan 24, 2018 the Blockport crowdsale successfully launched, completed, and was finalized, mitigating completely the risk of minting tokens freely as described below.
The crowdsale and tokens are considered to be in a stable state.*

~~Crowdsale owner can dilute investors' tokens by freely minting BPT:~~
~~BPT is a mintable token, meaning that for every crowdsale investor, the tokens they purchase are created upon purchase through the crowdsale contract. Because minting tokens is dangerous, the mint function is restricted to the BPT contract owner address only, which, logically, must be the crowdsale contract if it is to mint tokens for each purchase. The `resetTokenOwnership` function was added to the crowdsale contract in order to transfer token ownership to another address, in case the crowdsale's finalization function did not work as intended. While this makes sense from a safety perspective, transferring ownership of the BPT contract to the owner of the crowdsale poses a security risk for investors of the crowdsale, as it can occur mid-sale. This means that the owner of the crowdsale contract can call `resetTokenOwnership` mid-sale, mint tokens for themselves (or any other party), and transfer ownership back to the crowdsale contract for finalization. While tokens cannot be minted over the token supply cap, investors' tokens can be diluted at no cost in the event that the maximum~~

~~crowdsale cap is not reached.~~

~~The Authio team recommended removing the ability to use this function during the crowdsale. This issue was presented to the Blockport team along with our recommendation. Blockport has elected to keep the contract as is, with the understanding that this vulnerability exists, and will exist in the live contracts. The Blockport team has expressed that they are taking the responsibility of acting accordingly with the wishes of their investors.~~

3.5 Medium Severity

No medium severity issues were found.

3.6 Low Severity

Pre-Crowdsale token purchase rate change:
The crowdsale allows for the purchase rate of the token to be updated pre-crowdsale. However, given that the pre-sale sold out within minutes, it is likely that when the Blockport mainsale begins there will be a large number of people attempting to purchase immediately, without checking contract values and variables. Setting the token purchase rate directly before the crowdsale could have the effect that investors do not realize the rate they are purchasing at.

We recommend disallowing this function from being used within a few hours of crowdsale start.

3.7 Notes & Recommendations

Use the latest version of Solidity:
Most contract version pragmas were 0.4.13 or earlier. Using the latest version of Solidity (0.4.19, at time of writing), ensures that all the latest compiler updates and fixes, as well as language changes are reflected.

Use the latest version of OpenZeppelin contracts:
While OpenZeppelin template contracts are widely used and regarded to be safe, it is important to keep up with the latest versions available. The versions used in the tokensale repository also used Solidity 0.4.13 or earlier.

Using a token interface will reduce deployment costs:
Defining a `BlockportToken` variable means that the compiler adds the `BlockportToken.sol` bytecode to the crowdsale's bytecode at launch. Using an interface will cut down on this, reducing the gas cost necessary for deployment.

Creation of an un-investible crowdsale by mis-setting crowdsale cap:
It is possible to create an un-investible crowdsale by setting the crowdsale cap below the minimal investment amount. When a user attempts to purchase tokens, they cannot spend an amount that is both above the minimal investment amount, and below the crowdsale cap in wei. We recommend checking against this in the crowdsale constructor.

Use of SafeMath in order to adhere to best practices:
In several locations through the code, SafeMath is not used because the values being operated on could not possibly overflow. While this omission is acceptable, it would be wise to use SafeMath, simply to adhere to best practices.

Mark functions with appropriate visibilities:
In a few of the OpenZeppelin contracts, functions are not marked as `constant` or

`view`, where they should be. A complete list can be found in the line-by-line commented contracts, linked to in the Documents and Resources section.

Mark SafeMath library functions as `pure`:
SafeMath library functions do not modify or read state, and so should be marked as `pure`.

4. Documents & Resources

4.1 Line-By-Line Comments

Line-by-line commenting can be found in the EthereumAuthio github repository:

https://github.com/EthereumAuthio/Audits/tree/master/BlockportCrowdsale

4.2 Project Code

The current Blockport crowdsale and token code can be found in their public github repository:

https://github.com/Blockport/tokensale

5. Conclusion

The Authio team would like to commend the Blockport team on a well thought through project. Both the crowdsale and token contracts are high quality, thoroughly commented, and well organized. The contracts were simple to read and understand, and code function was clear from first glance. This, along with the other two independent contract audits and the bug bounty program, shows that the Blockport team has a very healthy attitude towards smart contract security.