Wibson Contract Audit
v2.2 RE-TEST

New Alchemy

August, 2018

Introduction

During August 2018, Wibson engaged New Alchemy to audit the Wibson smart contracts. The smart contracts implement functionality related to scheduling distribution of WIB tokens.

The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts, finding differences between the contracts' implementation and their behavior as described in public documentation, and finding any other issues with the contracts that may impact their trustworthiness. Wibson provided New Alchemy with access to the relevant source code and whitepapers.

The audit was performed over two days. This document describes the issues discovered in the audit.

UPDATE: Following delivery of this final version of the Wibson Smart Contract Security Audit Report, New Alchemy was asked to amend the report to address necessary changes to token name and contracts. Wibson Legal Counsel determined that the Token name would need to change. New Alchemy was informed of and confirmed that the only modifications were to the token contract name and the string token name.

The new commit is: `https://github.com/wibsonorg/tokens-distribution/tree/`

`Commit: 46eec4fb8ea32b573a3b00fa3a17dcc574808e56`

The token was renamed the following way:

Name of the Token: WIBSON

Symbol: WIB

Name of the smart contract: WIBToken

The decimal positions remain to be 9.

Note: The initial version of this document was provided to Wibson who then made various changes to their smart contract source code based upon New Alchemy's findings. This document now consists of the original and unchanged, other than typos and an updated inheritance chart, audit report (v1.0) overlaid with re-test results (v2.2). Re-test results are reflected in each issue title as 'Fixed', 'Partially Fixed', 'Not Fixed' or 'Informational'. Supporting re-test commentary is attached to a bolded 'Re-test v2.1:' prefix and placed at the end of the relevant section. The bulk of the re-test content relates to an examination of individual issues, along with brief comments in the executive summary and files audited sections. All figures are from the initial version of the document.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, compiler or tools stability, suitability of the business model, regulatory regime for the business model, or

any other statements about fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.

Executive Summary

The two smart contracts consisted of two source files, 53 passing tests and documentation. The contracts implement token vesting functionality. Overall they demonstrated use of established libraries, testing methods, and standard practices.

The audit identified issues with impacts ranging from moderate to very minor. The more significant findings involve

- Use of an outdated OpenZeppelin library.

While the test cases were not in scope, they were helpful to explore and confirm issues. The whitepaper and `README.md` were considered the primary documentation for intended functionality. A majority of the code was standard and copied from widely-used and reviewed contracts.

Re-test v2.1: New Alchemy has discussed the prior audit results with Wibson, advised on efficient and effective mitigation approaches, and inspected the resulting smart contract code provided for re-testing. New Alchemy has concluded:

- **All issues have been addressed and the associated risks effectively mitigated.**

Files Audited

The code reviewed by New Alchemy is in the GitHub repository https://github.com/wibsonorg/tokens-distribution at commit hash `98721c8fafd82d710d65fcb4b289a2f67790da9c`.

The specific files making up the complete deployed package include the following:

`Migrations.sol`	`29b7c1545eb35d96463fc04f4120cdb577446f7a`
`TokenTimelockPool.sol`	`24ac32b6aa95b4979b74cf970f352bb3c372fb5e`
`TokenVestingPool.sol`	`c647a4e6b656a1705fbb71629f74604d4c8bba8b`

https://github.com/OpenZeppelin/openzeppelin-solidity/tree/1.11.0

`contracts/math/SafeMath.sol`	`4b468f2d4ff3ff732c5618740319113bc3505828`
`contracts/ownership/Ownable.sol`	`2e3d8382f7f1d9872a263a980ecbb577b745123a`
`contracts/token/ERC20/ERC20Basic.sol`	`628c5212045740a0ba1f416934768bd6878e3cf6`
`contracts/token/ERC20/TokenTimelock.sol`	`e9446a9620550d998b59f98e0f57567cd1cd0890`
`contracts/token/ERC20/TokenVesting.sol`	`6283df769d3a64c347d2043b7084b620dd92badb`

Re-test v2.1: The revised contracts after the initial report was delivered are in commit a19173c9f5c98c601ad94a531e48e0b2668c645a.

New Alchemy's audit was additionally guided by Tokens.Distribution.for.Launch.pdf, shasum cbcba511a863aef4b560ccb67fea3628e017cc25.

General Discussion

The Wibson contracts primary function is to distribute tokens among beneficiaries using lock-up and vesting mechanisms using the standard OpenZeppelin contracts TokenTimelock[1] and TokenVesting[2].

The contracts use OpenZeppelin's SafeMath[3] which defines functions for safe math operations that will throw errors in the cases of integer overflow or underflows, OpenZeppelin's SafeERC20[4], and OpenZeppelin's Ownable[5] modifier.

The repository contained 53 truffle-based tests that ran and passed out of the box without requiring modifications. While New Alchemy generally utilizes the test cases to aid in inspecting and understanding the smart contract source code, they are outside of the primary security scope of the audit. Nonetheless, a large number of passing test cases is an excellent leading indicator of good quality.

[1] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/TokenTimelock.sol
[2] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/TokenVesting.sol
[3] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol
[4] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/SafeERC20.sol
[5] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/ownership/Ownable.sol

Contract Inheritance Diagram

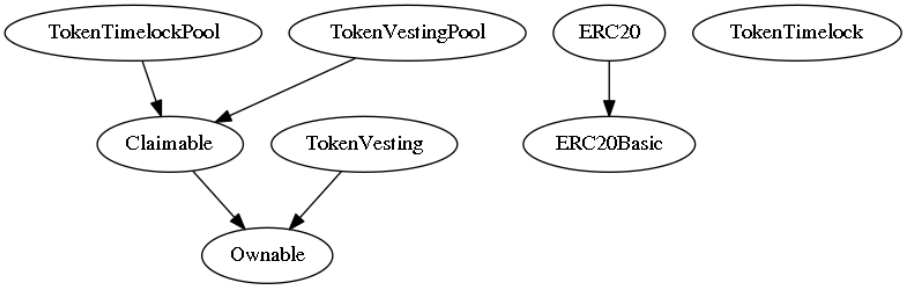

Figure 1: Contract Inheritance Diagram

Critical Issues

No critical issues were found.

Moderate Issues

A. *Fixed*: Outdated Dependencies

Re-test results: Wibson updated the OpenZeppelin dependency to 1.12.0.

Utilizing third-party, standardized, battle-tested components provides significant benefits in contract security. These components have a known track record and are generally carefully inspected by many people, well tested and quickly patched when vulnerabilities arise. However, many of these benefits are lost when dependencies become outdated or disconnected from their original source.

The `package.json` file present in the GitHub repository specifically includes the following dependency references:

```
"openzeppelin-solidity": "1.10.0",
```

Openzeppelin-solidity should be upgraded to 1.12.0 as follows:

```
"openzeppelin-solidity": "1.12.0",
```

Minor Issues

B. *Fixed*: State-changing functions do not emit events.

Re-test results: Wibson added events to all 3 cases.

The common security mantra of "deter, detect, delay and respond" provides a useful mental model for consideration during smart contract development. The 'detect' and 'respond' aspects are very dependent upon logging and auditing functionality. For these reasons, New Alchemy recommends emitting events whenever significant state-changing functionality is invoked (and further differentiating between success and failure) so auditing can be performed without a difficult direct inspection of the blockchain.

C. *Fixed*: Lack of Two-phase Ownership Transfer

Re-test results: Wibson now inherits the Claimable OpenZeppelin contract where appropriate.

In contracts that inherit the common `Ownable` contract from the OpenZeppelin project[6], a contract has a single owner. That owner can unilaterally transfer ownership to a different address. However, if the owner of a contract makes a mistake in entering the address of an intended new owner, then the contract can become irrecoverably unowned. This is analogous to the well-known "Locked Ether" scenario resulting in several million US dollars (over 7000 Ether) irretrievably locked at address 0x0[7].

In order to prevent this high-impact scenario, New Alchemy recommends implementing a two-phase ownership transfer. In this model, the original owner designates a new owner but does not actually transfer ownership. The new owner then accepts ownership and completes the transfer. This can be implemented as follows:

```
contract Ownable {
    address public owner;
    address public newOwner

    event OwnershipTransferred(address indexed oldOwner, address indexed newOwner);

    function Ownable() public {
    owner = msg.sender;
    newOwner = address(0);
    }

    modifier onlyOwner() {
    require(msg.sender == owner);
    _;
    }
```

[6] https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/ownership/Ownable.sol
[7] https://etherscan.io/address/0x0000000000000000000000000000000000000000

```
16
17      function transferOwnership(address _newOwner) public onlyOwner {
18      require(address(0) != _newOwner);
19      newOwner = _newOwner;
20      }
21
22      function acceptOwnership() public {
23      require(msg.sender == newOwner);
24      OwnershipTransferred(owner, msg.sender);
25      owner = msg.sender;
26      newOwner = address(0);
27      }
28  }
```

Smart Contract ABI Reference

This section provides the fully-elaborated application binary interface (ABI) for the primary contracts as seen at runtime. The ABI defines how the contracts may be interacted with while running inside the EVM. The tables make for very easy inspection of top-level specifics including:

- The definitive reference of all exposed functions
- Correct input and output function signatures
- Presence of constructors and fallbacks as expected
- Correct state mutability (pure, view, nonpayable, payable)
- Extraneous functions which should be private

`contracts/TokenTimelockPool.sol` ABI Reference

Function	Inputs Type:Name	Outputs Type:Name	State Mutability
addBeneficiary	address:_beneficiary uint256:_amount	address:–	nonpayable
beneficiaries	uint256:–	address:–	view
beneficiaryDistributionContracts	address:– uint256:–	address:–	view
constructor()	address:_token uint256:_totalFunds uint256:_releaseDate	-NONE-	nonpayable
distributedTokens	-NONE-	uint256:–	view
getDistributionContracts	address:_beneficiary	address[]:–	view
owner	-NONE-	address:–	view
reclaim	-NONE-	bool:–	nonpayable
releaseDate	-NONE-	uint256:–	view
renounceOwnership	-NONE-	-NONE-	nonpayable
token	-NONE-	address:–	view
totalFunds	-NONE-	uint256:–	view
transferOwnership	address:_newOwner	-NONE-	nonpayable

`contracts/TokenVestingPool.sol` ABI Reference

Function	Inputs Type:Name	Outputs Type:Name	State Mutability
addBeneficiary	address:_beneficiary uint256:_start uint256:_cliff uint256:_duration uint256:_amount	address:–	nonpayable
beneficiaries	uint256:–	address:–	view
beneficiaryDistributionContracts	address:– uint256:–	address:–	view
constructor()	address:_token uint256:_totalFunds	-NONE-	nonpayable
distributedTokens	-NONE-	uint256:–	view
getDistributionContracts	address:_beneficiary	address[]:–	view
owner	-NONE-	address:–	view
renounceOwnership	-NONE-	-NONE-	nonpayable
token	-NONE-	address:–	view
totalFunds	-NONE-	uint256:–	view
transferOwnership	address:_newOwner	-NONE-	nonpayable

Line by line comments

This section lists comments on design decisions and code quality made by New Alchemy during the review. They are not known to represent security flaws.

A. Use of `now`

`contracts/TokenTimelockPool.sol` Lines 63, 93, 129

Solidity coding style convention prefers the explicit use of `block.timestamp` over its alias of `now` to more clearly reflect the dependence upon block mining. Also, note that this value can be manipulated by the block miner by approximately 30 seconds. It does not appear that fundamental contract integrity is dependent upon precise timestamps, but this vulnerability should be kept in mind as the code evolves.

B. *Fixed*: Emit events when changing state

Re-test results: Wibson now emits events in all 3 cases.

`contracts/TokenTimelockPool.sol` Lines 86, 127

`contracts/TokenVestingPool.sol` Line 95

State changing functions should emit events.