ArcadierX Audit

March 1st, 2019

By Coinfabrik

Introduction 3

Summary 3

Detailed findings 4

Minor severity 4

No Solidity pragma 4

Solidity errors on newer compiler version 4

Inline assembly should be encapsulated in a function 4

Enhancements 5

Insufficient documentation 5

Commented code 5

Typo in variable name 5

Repeated code in transfer and transferFrom 5

Notes 5

Conclusion 6

Introduction

CoinFabrik was asked to audit the contracts for the ArcadierX project. Firstly, we will provide a summary of our discoveries and secondly, we will show the details of our findings.

Summary

The contracts audited are from the Arcadierx repository at https://github.com/arcadierx/arcadierx. The audit is based on the commit 985f836527544ea3878f21d2ee68c47e079ccdd4.

The audited contracts are:

- IReceiver: Defines an interface for contracts that are callable after receiving tokens.
- LSafeMath: Overflow checked arithmetic operations.
- Ownable: Common privilege function modifier.
- ERC20Basic: ERC20 Token interface with only transfer and balance.
- BasicToken: Implements ERC20Basic interface using IReceiver to call receiving contracts.
- ERC20: ERC20 Token full interface.
- StandardToken: Fully implements the ERC20 token interface using IReceiver to call receiving contracts.
- ARCXToken: The final implementation details of the token and entry points are specified in this contract. This is the one that gets deployed in the blockchain.

No issues of critical, high or medium severity were found.

The following analyses were performed:

- Misuse of the different call methods: call.value(), send() and transfer().
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Old compiler version pragmas.
- Race conditions such as reentrancy attacks or front running.
- Misuse of block timestamps, assuming anything other than them being strictly increasing.
- Contract softlocking attacks (DoS).
- Potential gas cost of functions being over the gas limit.
- Missing function qualifiers and their misuse.
- Fallback functions with a higher gas cost than the one that a transfer or send call allows.
- Fraudulent or erroneous code.

- Code and contract interaction complexity.
- Wrong or missing error handling.
- Overuse of transfers in a single transaction instead of using withdrawal patterns.
- Insufficient analysis of the function input requirements.

Detailed findings

Minor severity

No Solidity pragma

The contract code given does not have a Solidity version pragma. This is typically expected of contract code. Adding the pragma helps to know the compiler version that is being used in the project. The code itself doesn't compile on version 0.5.0 or later which further reinforces this issue.
We recommend adding a Solidity pragma to address this issue.

Solidity errors on newer compiler version

Solidity errors because of the usage of the obsolete qualifier *constant.* The contracts also have the old constructor declaration and do not explicitly define the data location on array-typed parameters which is required in version 0.5.0. Using old Solidity versions is not advised as newer versions address problems and bugs older versions may have, especially considering the immaturity of the compiler at the time of writing this document.
We recommend migrating to a newer version of the Solidity compiler.

Inline assembly should be encapsulated in a function

There is inline assembly in both transfer functions to check whether an address is a contract:

```
    assembly {
            // Retrieve the size of the code on target address, this
needs assembly .
            codeLength := extcodesize(_to)
    }
```

Code like this should be encapsulated in a function to prevent errors. Assembly code is much more dangerous when called in the middle of functions as they have access to all the context inside them.
We recommend moving this code to a single function.

Enhancements

Insufficient documentation

The functions in the contract ARCXToken are not documented. Having these documented would help knowing the functionality expected from them.
We recommend documenting these functions using the NatSpec found in the other contracts documented.
There is also no documentation regarding the reentrancy of the contracts allowed by IReceiver (ERC223) interface. We also recommend documenting this behavior as it may allow for vulnerabilities if the code is modified in the future.

Commented code

There is commented code in the *transferFrom* function:

```
//code changed to comply with ERC20 standard
balances[_from] = balances[_from].sub(_value);
balances[_to] = balances[_to].add(_value);
//balances[_from] = balances[_from].sub(_value); // this was removed
allowed[_from][msg.sender] = _allowance.sub(_value);
```

Commented code is a problem as it clutters and weakens the main intention of the code.
We recommend removing the commented code.

Typo in variable name

There is a typo in the variable *ingnoreLocks* at ARCXToken. It should be called *ignoreLocks.* We recommend fixing typos as they weaken the readability of the contracts.

Repeated code in transfer and transferFrom

The *transfer* and *transferFrom* implementations have repeated code that can be reused. It is common to have a private function implementing the repeated code, and having these two functions call that instead. Repeated code increases the surface area for errors so eliminating it should be considered when possible.

Notes

The token allows reentrancy since it calls the receiving contracts when transferring tokens. However, the calls are made at the end of each function after modifying all the storage. This avoids exposing an unexpected contract

invariant in any of the calls, and thus makes them safe as the contract isn't in an invalid state if reentrancy occurs.

Conclusion

This is a simple token. It does not have many features so it has little surface for errors. The contracts were readable and overall easy to follow. No issues of critical, high or medium severity were found. There are only a couple of minor issues: not using nor expliciting a recent compiler, and the use of unencapsulated inline assembly. Both of them can be fixed by small updates to the contracts. These issues impact future development mostly, and thus, the decision of fixing them is left to the developers. This kind of issue does not warrant a redeployment of the contracts.

Finally, we examined the contract bytecode deployed at the Ethereum mainnet address *0x1A506CBc15689F2902b22A8baF0e5Cc1eD8203eE*. We found it to match with the bytecode produced by the Solidity compiler v0.4.25 when compiling the contracts.