Security Audit Report: LaborCyptoToken Smart Contracts (Update)

An audit is performed on the LaborCryptoToken smart contract; the code is written in *Solidity* and makes consistent use of the community recommended methods defined in the *OpenZeppelin* library of ERC20 smart contracts.

This audit makes use of the *Truffle* testing framework for unit tests; the *Mythril* testing suite for on-chain interactions and *SmartCheck* online utility to test against known attack vectors.

An analysis of the implementation and security recommendations follow:

Critical Severity

No issues of critical severity.

High Severity

No issues of high severity.

Medium Severity

Vulnerability to Short Address Exploits

This is a recently discovered vulnerability where an attacker may indirectly manipulate the transfer amount by making use of an ethereum address that ends with one or more zeroes. Crucially, this tends to occur after internal sanity checks have passed. While not a commonly used attack vector, it remains prudent to take preventative measures against such.

Recommendation:
Consider using a modifier in the *transfer* and *approve* methods that checks for a minimum transaction data size.

Update: *onlyPayloadSize* modifier has now been implemented; vulnerability to short address attack is eliminated.

Low Severity

Locking the *Pragma Solidity* compiler version

By locking the compiler version we preclude any vulnerabilities that may arise in future releases of the compiler.

Recommendation:
Consider locking the compiler version to *0.4.23*

Update: Solidity version is locked to *0.4.23*

Prevent token transfers to the token contract address

Without this extra check, it becomes possible for a token holder to erroneously transfer their tokens to the token smart contract itself - from where it may not be recovered.

Recommendation:
Consider checking within the transfer method that the *_to* address is not the token contract address.

Update: *transfer* and *transferFrom* methods now check that *_to* address is not the token contract itself

Notes & Additional Information

- An additional recommendation is to emit a *Transfer* event when the *totalSupply* is first assigned to the contract *owner*. This follows from the ERC20 protocol.
- For better readability it is recommended that each function and variable viewability specifier e.g: *public, private, external,* etc are explicitly stated in the body of he code.
- Consider employing a *fallback* function that will revert any Ether erroneously sent to the contract. Alternatively, one could employ a function to extract such Ether and other erroneously sent ERC20 tokens that are trapped in the contract.
- Since it has been suggested that a crowdsale smart contract might not be used - it might be prudent to employ a *batchTransfer* function in order to distribute tokens to multiple parties.

Update: *Transfer* event is emitted on contract deployment
Viewability specifiers have been explicitly declared
Fallback function reverts incoming erroneous Ether transactions
batchTransfer method is implemented without any security concern

Conclusion

No critical or high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Update: Recommendations were accepted and implemented satisfactorily.