DSLA TOKEN SMART CONTRACT AUDIT

FOR STACKTICAL SAS

15.11.2018

Made in Germany by Chainsulting.de

Smart Contract Audit DSLA Token

Table of Contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, 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.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Stacktical SAS. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description	Author
0.1 (28.10.2018)	Layout	Y. Heinze
0.5 (29.10.2018)	Automated Security Testing	Y. Heinze
0.7 (30.10.2018)	Manual Security Testing	Y. Heinze
1.0 (30.10.2018)	Summary and Recommendation	Y. Heinze
1.5 (15.11.2018)	Deploy to Main Network Ethereum	Y. Heinze
1.6 (15.11.2018)	Last Security Check and adding of recommendations	Y. Heinze
1.7 (15.11.2018)	Updated Code Base	Y. Heinze

2. About the Project and Company

Company address:

STACKTICAL SAS
3 BOULEVARD DE SEBASTOPOL
75001 PARIS FRANCE

RCS 829 644 715
VAT FR02829644715

Project Overview:

Stacktical is a french software company specialized in applying predictive and blockchain technologies to performance, employee and customer management practices.

Stacktical.com is a comprehensive service level management platform that enables web service providers to automatically indemnify consumers for application performance failures, and reward employees that consistently meet service level objectives.

Company Check:
https://www.infogreffe.fr/entreprise-societe/829644715-stacktical-750117B117250000.html

3. Vulnerability Level

0-Informational severity – A vulnerability that have informational character but is not effecting any of the code.

1-Low severity - A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

2-Medium severity – A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

3-High severity – A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.

4-Critical severity – A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.

4. Overview of the audit

The DSLA Token is part of the DSLA Crowdsale Contract and both where audited. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work.

4.1 Used Code from other Frameworks/Smart Contracts (3th Party)

1. SafeMath.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol

2. ERC20Burnable.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/ERC20Burnable.sol

3. ERC20.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/ERC20.sol

4. IERC20.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/IERC20.sol

5. Ownable.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/ownership/Ownable.sol

6. Pausable.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/lifecycle/Pausable.sol

7. PauserRole.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/access/roles/PauserRole.sol

8. Roles.sol
https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/access/Roles.sol

4.2 Tested Contract Files

File	Checksum (SHA256)
contracts\Migrations.sol	700c0904cfbc20dba65f774f54a476803a624372cc95d926b3eba9b4d0f0312e
contracts\DSLA\DSLA.sol	00339bccbd166792b26a505f10594341477db5bcd8fe7151aebfe73ac45fbb9f
contracts\DSLA\LockupToken.sol	3afc805367c072082785f3c305f12656b0cbf1e75fe9cfd5065e6ad3b4f35efa
contracts\Crowdsale\ DSLACrowdsale.sol	fc66d9d53136278cf68d7515dc37fab1f750cacff0079ceac15b175cf97f01bc
contracts\Crowdsale\Escrow.sol	2c912e901eb5021735510cb14c6349e0b47d5a3f81d384cfb1036f1f0e30996c
contracts\Crowdsale\ PullPayment.sol	a5a98901913738df2ff700699c2f422944e9a8a270c708fdbc79919fc9b30a42
contracts\Crowdsale\ VestedCrowdsale.sol	e6b70f3dfd294e97af28ecd4ee720ffbf1ca372660302e67464ecf69d4da6a9b
contracts\Crowdsale\ Whitelist.sol	027aa5a6799bd53456adfb4ef9f0180890a376eeeb4c6ae472388af6ea78b308

4.3 Contract Specifications (DSLA Token)

Language	Solidity
Token Standard	ERC20
Most Used Framework	OpenZeppelin
Compiler Version	0.4.24
Burn Function	Yes (DSLACrowdsale.sol)
Mint Function	Yes
Ticker Symbol	DSLA
Total Supply	10 000 000 000
Timestamps used	Yes (Blocktimestamp in DSLACrowdsale.sol)

5. Summary of Contracts and Methods

Functions will be listed as:

[Pub] public
[Ext] external
[Prv] private
[Int] internal

A ($)denotes a function is payable.
A # indicates that it's able to modify state.

5.1 DSLA Token

Shows a summary of the contracts and methods

+ DSLA (LockupToken)
  - [Pub] <Constructor> #

+ LockupToken (ERC20Burnable, Ownable)
  - [Pub] <Constructor> #
  - [Pub] setReleaseDate #
  - [Pub] setCrowdsaleAddress #
  - [Pub] transferFrom #
  - [Pub] transfer #
  - [Pub] getCrowdsaleAddress

+ Ownable
  - [Int] <Constructor> #
  - [Pub] owner
  - [Pub] isOwner
  - [Pub] renounceOwnership #
  - [Pub] transferOwnership #
  - [Int] _transferOwnership #

5.2 Crowdsale
Shows a summary of the contracts and methods

+ DSLACrowdsale (VestedCrowdsale, Whitelist, Pausable, PullPayment)
  - [Pub] <Constructor> #
  - [Ext] <Fallback> ($)
  - [Pub] buyTokens ($)
  - [Pub] goToNextRound #
  - [Pub] addPrivateSaleContributors #
  - [Pub] addOtherCurrencyContributors #
  - [Pub] closeRefunding #
  - [Pub] closeCrowdsale #
  - [Pub] finalizeCrowdsale #
  - [Pub] claimRefund #
  - [Pub] claimTokens #
  - [Pub] token
  - [Pub] wallet
  - [Pub] raisedFunds
  - [Int] _deliverTokens #
  - [Int] _forwardFunds #
  - [Int] _getTokensToDeliver
  - [Int] _handlePurchase #
  - [Int] _preValidatePurchase
  - [Int] _getTokenAmount
  - [Int] _doesNotExceedHardCap
  - [Int] _burnUnsoldTokens #

+ Escrow (Ownable)
  - [Pub] deposit ($)
  - [Pub] withdraw #
  - [Pub] beneficiaryWithdraw #
  - [Pub] depositsOf

+ PullPayment
  - [Pub] <Constructor> #
  - [Pub] payments
  - [Int] _withdrawPayments #
  - [Int] _asyncTransfer #
  - [Int] _withdrawFunds #

+ VestedCrowdsale
  - [Pub] getWithdrawableAmount
  - [Int] _getVestingStep
  - [Int] _getValueByStep

+ Whitelist (Ownable)
  - [Pub] addAddressToWhitelist #
  - [Pub] addToWhitelist #
  - [Pub] removeFromWhitelist #

6. Test Suite Results (DSLA Token)

6.1 Mythril Classic Security Audit
Mythril Classic is an open-source security analysis tool for Ethereum smart contracts. It uses concolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities.

```
ubuntu@testing:~$ myth -x stacktical-tokensale-contracts/contracts/DSLA/DSLA.sol
WARNING:root:No contract was created during the execution of contract creation Increase the resources for creation execution (--max-depth or --create-timeout)
The analysis was completed successfully. No issues were detected.
```

Result: The analysis was completed successfully. No issues were detected.

6.2 Oyente Security Audit
Oyente is a symbolic execution tool that works directly with Ethereum virtual machine (EVM) byte code without access to the high level representation (e.g., Solidity, Serpent).

Result: The analysis was completed successfully. No issues were detected

7. Test Suite Results (Crowdsale)

7.1 Mythril Classic Security Audit
Mythril Classic is an open-source security analysis tool for Ethereum smart contracts. It uses concolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities.

```
ubuntu@testing:~$ myth -x stacktical-tokensale-contracts/contracts/Crowdsale/DSLACrowdsale.sol
WARNING:root:No contract was created during the execution of contract creation Increase the resources for creation execution (--max-depth or --create-timeout)
The analysis was completed successfully. No issues were detected.
```

Result: The analysis was completed successfully. No issues were detected.

7.2 Oyente Security Audit
Oyente is a symbolic execution tool that works directly with Ethereum virtual machine (EVM) byte code without access to the high level representation (e.g., Solidity, Serpent).

Result: The analysis was completed successfully. No issues were detected

8. Specific Attacks (DSLA Token & Crowdsale)

Attack	Code Snippet	Severity	Result/Recommendation

An Attack Vector on Approve/TransferFrom Methods
 	
Source:
https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
	In file: openzeppelin-solidity-master\contracts\token\ERC20\ERC20.sol:74-80
	`function approve(address spender, uint256 value) public returns (bool) {`
	`  require(spender != address(0));`
	
	`  _allowed[msg.sender][spender] = value;`
	`  emit Approval(msg.sender, spender, value);`
	`  return true;`
	`}`
		Severity: 2	Only use the approve function of the ERC-20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).
			The DSLA Smart Contract is secure against that attack

Timestamp Dependence "block.timestamp" can be influenced by miners to a certain degree. **Source:** https://smartcontractsecurity.github.io/SWC-registry/docs/SWC-116	In file: stacktical-tokensale-contracts-master\contracts\DSLA\LockupToken.sol:23 `require(_releaseDate > block.timestamp);`	**Severity: 0**	Developers should write smart contracts with the notion that block timestamp and real timestamp may vary up to half a minute. Alternatively, they can use block number or external source of timestamp via oracles.
Unchecked math: Solidity is prone to integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account.	No critical mathematical functions are used	**Severity: 2**	Check against over- and underflow (use the SafeMath library). The DSLA Smart Contract is secure against that attack

Unhandled Exception A call/send instruction returns a non-zero value if an exception occurs during the execution of the instruction (e.g., out-of-gas). A contract must check the return value of these instructions and throw an exception.		**Severity: 0**	Catching exceptions is not yet possible.
Sending tokens (not Ethereum) to a Smart Contract It can happen that users without any knowledge, can send tokens to that address. A Smart Contract needs to throw that transaction as an exception.		**Severity: 1**	The function of sending back tokens that are not whitelisted, is not yet functional. The proposal ERC223 can fix it in the future. https://github.com/Dexaran/ERC223-token-standard
SWC ID: 110 A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds array access, or assert violations. This is acceptable in most situations. Note however that 'assert()' should only be used to check invariants. Use	In file: stacktical-tokensale-contracts/contracts/Crowdsale/Escrow.sol :40 `assert(address(this).balance >= payment)`	**Severity: 1**	The DSLA Smart Contract is secure against that exception

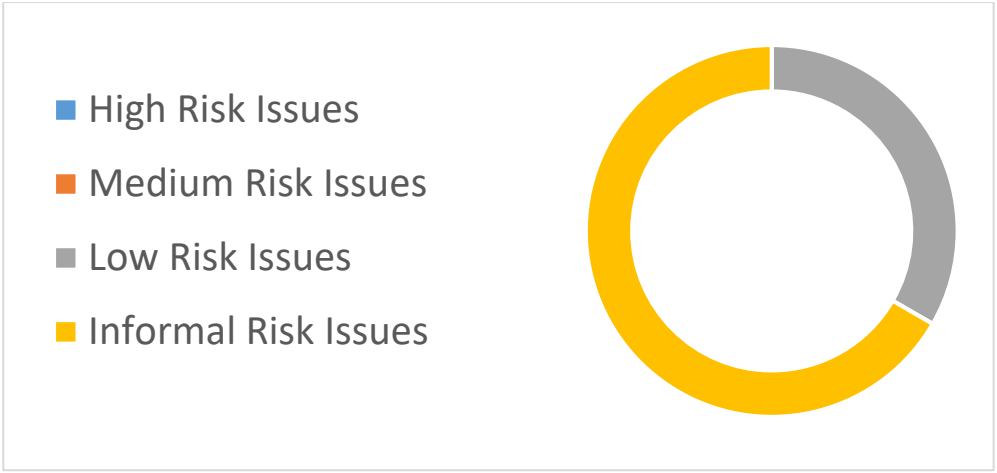
‘require()’ for regular input checking.			

Sources:

https://smartcontractsecurity.github.io/SWC-registry
https://dasp.co
https://github.com/ChainsultingUG/solidity-security-blog
https://consensys.github.io/smart-contract-best-practices/known_attacks

9. Executive Summary

A majority of the code was standard and copied from widely-used and reviewed contracts and as a result, a lot of the code was reviewed before. It correctly implemented widely-used and reviewed contracts for safe mathematical operations. The audit identified no major security vulnerabilities, at the moment of audit. We noted that a majority of the functions were self-explanatory, and standard documentation tags (such as @dev, @param, and @returns) were included.



10. General Summary

The issues identified were minor in nature, and do not affect the security of the contract.

Additionally, the code implements and uses a SafeMath contract, which defines functions for safe math operations that will throw errors in the cases of integer overflow or underflows. The simplicity of the audited contracts contributed greatly to their security. The usage of the widely used framework OpenZeppelin, reduced the attack surface.

11. Deployed Smart Contract

https://etherscan.io/address/0x8efd96c0183f852794f3f18c48ea2508fc5dff9e (Crowdsale)

https://etherscan.io/address/0xEeb86b7c0687002613Bc88328499F5734e7Be4c0 (DSLA Token)

We recommended to Update the etherscan.io information with Logo/Website/Social Media Accounts (DSLA Token) and verify the Smart Contract Code (Both Contracts). That gives buyers more transparency.

Updated Code Base After Audit (No Impairment)

Readjust of caps:

https://github.com/Stacktical/stacktical-tokensale-contracts/pull/6/files

Token burn optional:

https://github.com/Stacktical/stacktical-tokensale-contracts/pull/3/files