SmartDec

Digitex Vesting Smart Contracts Security Analysis

This report is public.

Published: February 15, 2018

Abstract

In this report we consider the security of the Digitex Vesting project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclosure

The contract in consideration was developed by SmartDec, this is our internal report. We follow our usual procedure for security audit of a smart contract (described below), and present the report in the same manner as for an external audit.

Procedure

In our analysis we consider Digitex Vesting whitepaper (Digitex-Whitepaper.v.1.1.pdf, sha1sum c0d1808fce894e864f688ab40eb5c0b2601c836a), documentation and smart contracts code (version on commit cd77395).
We perform our audit according to the following procedure:

- automated analysis
  - we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
  - we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, Solhint, and Securify (beta version since full version was unavailable at the moment this report was made)
  - we manually verify (reject or confirm) all the issues found by tools
- manual audit
  - we manually analyze smart contracts for security vulnerabilities
  - we check smart contracts logic and compare it with the one described in the documentation
  - we run tests and check code coverage
  - we deploy contracts to testnet and test them manually
- report
  - we reflect all the gathered information in the report

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Checked vulnerabilities

We have scanned Digitex Vesting smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DOS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

About The Project

Project Architecture

For the audit, we have been provided with the following set of files:

- ERC223.sol (with ERC223 interface)
- ERC223ReceivingContract.sol (with ERC223ReceivingContract interface)
- Vesting.sol (with Vesting contract, which inherits ERC223ReceivingContract interface and Ownable contract from OpenZeppelin library, version 1.6.0)

Total size of the contracts is 749 lines of code.
Provided file set is a truffle project and an npm package. The project is compiled with a combination of `npm install` (installs all dependencies, including the OpenZeppelin library) and `npm run truffle compile` commands.
The project compiles successfully without warnings (see the compilation output in Appendix).
The project contains tests that are run with the `npm run test` command and these tests successfully pass. Test coverage is full (see the output of the test command and code coverage in the Appendix).
The project contains deploy scripts for the Vesting contract. The project contains directory with building and testing scripts. The project also contains configuration files for the solium linter, solidity-coverage and doxity tools.

Code Logic

The code logic is fully consistent with the documentation.
ERC223 token interface is ERC223 compatible, ERC223ReceivingContract is interface with `takenFallback` function to handle token transfers to the contract (ERC223 standard compliance was verified in the audit).
Vesting contract implements vesting of 100 000 000 DGTX tokens:

1. Owner of the contract is the address the contract is deployed from.
2. The owner can transfer its ownership to another address.
3. The parameter of the constructor is contract address (deploy script passes the address of DGTX token as parameter).
4. `tokenFallback` function implements token receiving: only 100 000 000 tokens can be received and tokens can be received only once.
5. Owner of the contract can withdraw their tokens according to the following scheme: on July 15 2018 25% of tokens locked in the contract become available; further, after every quarter of the year the contract unlocks 6.25% of the tokens it is holding.

Automated Analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of their analyses. All the issues found by tools were manually checked (rejected or confirmed).

Tool	Vulnerability	False positives	True positives
Remix	Gas requirement of function high	4	
	Potential Violation of Checks-Effects-Interaction pattern	2	
	Use assert(x) / require(x)	1	
	use of "now"		2
Total Remix		7	2
Securify*	-		
Total Securify*			
SmartCheck	No Payable Fallback		2
	Reentrancy External Call	4	
	Unchecked Math	8	
Total SmartCheck		12	2
Solhint	Avoid to make time-based decisions in your business logic		2
	Event and function names must be different	2	
Total Solhint		2	2
Overall Total		21	6

Securify* — beta version, full version is unavailable.

Cases where these issues lead to actual bugs or vulnerabilities are described in the next section.

Manual Analysis

Contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. Besides, the results of automated analysis were manually verified. All confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

The audit showed no medium severity issues.

Low severity issues

Low severity issues can influence smart contracts operation in future versions of code. We recommend to take them into account.

The audit showed no low severity issues.

Conclusion

In this report we considered the security of Digitex Vesting smart contract. We performed our audit according to the procedure described above.
The audit showed high code quality, high test coverage, and no security issues.

This analysis was performed by SmartDec.

COO Sergey Pavlin 

February 15, 2018

Appendix

Code coverage

```
"--------------|----------|----------|----------|----------|--
--------------|
File          |  % Stmts | % Branch |  % Funcs |  % Lines
|Uncovered Lines |
--------------|----------|----------|----------|----------|---
-------------|
 contracts/   |      100 |      100 |      100 |      100
|                |
  Vesting.sol |      100 |      100 |      100 |      100
|                |
--------------|----------|----------|----------|----------|---
-------------|
All files     |      100 |      100 |      100 |      100
|                |
--------------|----------|----------|----------|----------|---
-------------|"
```

Compilation output

```
"Compiling .\contracts\ERC223.sol...
Compiling .\contracts\ERC223ReceivingContract.sol...
Compiling .\contracts\Migrations.sol...
Compiling .\contracts\Vesting.sol...
Compiling zeppelin-solidity/contracts/ownership/Ownable.sol...
Writing artifacts to .\build\contracts"
```

Tests output

```
"Contract: Vesting
    √ Checking if important functions are unavailable for no
owner (177ms)
    √ Check ownership transfer (209ms)
    √ Checking wrong fallback call (393ms)
```

```
    √ Checking normal work process (1316ms)
    √ Checking withdraw all (229ms)
5 passing (2s)"
```

Solhint output

```
"contracts/ERC223.sol
  15:5  warning  Event and function names must be
different  no-simple-event-func-name

contracts/Vesting.sol
  45:5   warning  Event and function names must be
different                 no-simple-event-func-name
  68:13  warning  Avoid to make time-based decisions in your
business logic  not-rely-on-time
  72:26  warning  Avoid to make time-based decisions in your
business logic  not-rely-on-time

✖ 4 problems (4 warnings)"
```

Solium output

```
"contracts\Vesting.sol
  68:12    warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  72:25    warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members

✖ 2 warnings found."
```