SmartDec

Digitex Auction Smart Contracts Security Analysis

This report is public.

Published: February 15, 2018

Abstract

In this report we consider the security of the Digitex Auction 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 Auction documentation (version on commit e51a834) and smart contracts code (version on commit e51a834).
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 Auction 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:

- Auction.sol (with Auction contract, which inherits ERC223ReceivingContract interface, usingOraclize contract from Oraclize library and Ownable contract from OpenZeppelin library, version 1.6.0)
- ERC223.sol (with ERC223 interface)
- ERC223Receiver.sol (with ERC223ReceivingContract interface)

Total size of the contracts is 1436 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 main contract compiles successfully without warnings (see the compilation output in Appendix, all warnings are for usingOraclize contract).

The project contains tests that are run with the `npm run test` command and successfully pass. Tests coverage is full (see the output of the test command and code coverage in the Appendix, only oraclize functions are not covered).

The project contains deploy scripts for the Auction 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.

Auction contract implements Dutch auction for 100 000 000 DGTX tokens.

Token auction will start at 9am EST on February 15th, 2018 (this date is passed to the contract constructor in deploy script). The price starts from $0.25 per 1 DGTX and linearly decreases by $0.01 every hour until it reaches $0.01. The auction lasts no more than 30 days.

Public can participate in auction by sending ETH to the auction contract. It is not allowed to participate from contract accounts, only from external accounts.

Buyer's bid is calculated in USD cents according to the ETH/USD exchange rate in the contract at the time ETH are received.

Contributions are limited: maximum - $10000 per address (this value is passed to the contract constructor in deploy script), minimum - 0.01 ETH per transaction. The ETH/USD rate is updated every hour using an oracle. To launch rate update cycle someone needs to call `startEthToCentsRateUpdateCycle` function.

Owner of the contract can transfer and withdraw ETH to/from the contract, this is needed to pay commission to the oracle.
Auction is finished when the amount of USD received (sum of all bids) is equal to the current valuation of the tokens offered (that is, current auction token price multiplied by total amount of tokens offered).
After the auction is finished, buyers can claim tokens according to the final auction price and the size of their bid. Each buyer receives the portion of all tokens offered proportional to their bid (calculated in USD at the moment ETH are received as described above). This way each buyer receives tokens according to the final price at the moment of the end of auction. Each buyer is guaranteed to receive tokens at a price no higher than the price at which they bid.
Unsold token are transferred to the owner.
Anyone can trigger the claiming of tokens for some addresses using `distributeTokensRange` function.
Extra ETH (above the maximum per address or what is needed to close the auction) are sent back to the bidder.

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	Fallback function of contract requires too much gas		1
	Gas requirement of function	33	
	Is constant but potentially should not be	1	
	Potential Violation of Checks-Effects-Interaction pattern	1	
	Variables have very similar names	1	1
Total Remix		36	2
Securify*	Transactions May Affect Ether Receiver	4	
	Transactions May Affects Ether Amount		4
Total Securify*		4	4
Solhint	Avoid to make time-based decisions in your business logic		5
	Avoid to use inline assembly		1
	Event and function names must be different	1	
	Fallback function must be simple		1
Total Solhint		1	7
SmartCheck	Costly loop		1
	Reentrancy	1	

	The incompleteness of the compiler, view-function		1
	Unchecked math	34	
Total SmartCheck		35	2
Overall Total		76	15

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.

Large fallback function

The Auction.sol fallback function is large. If the fallback function requires more than 2300 gas, the contract cannot receive ETH through some of standard ways. We recommend to avoid creating large fallback functions. However, since in this case this is necessary, we recommend to mention in the API documentation that contract might not be able to accept ETH in some cases.

Conclusion

In this report we have considered the security of Digitex Auction smart contract. We performed our audit according to the procedure described above.
The audit showed high code quality, high test coverage, and no critical or medium severity issues. We recommend mentioning Large fallback function low severity issue in API documentation.

This analysis was performed by SmartDec.

COO Sergey Pavlin [illegible]

February 15, 2018

Appendix

Code coverage

```
--------------|----------|----------|----------|----------|---
-------------|
File | % Stmts | % Branch | % Funcs | % Lines |Uncovered Lines
|
--------------|----------|----------|----------|----------|---
-------------|
contracts/ | 84.81 | 84.38 | 81.25 | 87.21 | |
Auction.sol | 84.81 | 84.38 | 81.25 | 87.21 |... 158,282,283 |
--------------|----------|----------|----------|----------|---
-------------|
All files | 84.81 | 84.38 | 81.25 | 87.21 | |
--------------|----------|----------|----------|----------|---
-------------|
```

Compilation output

```
$ truffle compile --reset
Compiling .\contracts\Auction.sol...
Compiling .\contracts\ERC223.sol...
Compiling .\contracts\ERC223ReceivingContract.sol...
Compiling .\contracts\oraclize\OraclizeAddrResolverI.sol...
Compiling .\contracts\oraclize\OraclizeI.sol...
Compiling .\contracts\oraclize\usingOraclize.sol...
Compiling zeppelin-solidity/contracts/ownership/Ownable.sol...

(only usingOraclize)

Writing artifacts to .\build\contracts
```

Tests output

```
Contract: Auction
```

```
construction
√ Check 0x0 token
√ Check 0x0 beneficiary
√ Check startTime <= now (152ms)
√ Check maxBid = 0 (167ms)
√ Check ownership transfer (192ms)
√ Checking initialized (380ms)
Initial state
Checking Tokensale accept tokens in initial state
√ Can't call not from token.
√ Can't transfer not 100 000 000 tokens. (42ms)
√ Can transfer tokens to auction in initial state. (69ms)
√ Can't transfer tokens twice. (84ms)
√ Can transfer not 100 000 000 tokens after the first
transfer. (76ms)
Checking one's restriction
√ Checking that one can't bid or withdraw before auction
√ Checking that one can't bid from contract (83ms)
Main stage
√ Checking owner can send ether and withdraw it (2012ms)
√ Checking can't send less than transaction minimum (227ms)
√ Checking can't invest more than address maximum (280ms)
Finising
√ Checking buying tokens and finish by user payment (605ms)
√ Checking buying tokens and finish by user payment (625ms)
√ Checking buying tokens and finish by price decrease (331ms)
√ Checking finishing auction by time (469ms)

20 passing (20s)
```

Solhint output

```
contracts/Auction.sol
   59:30  warning  Avoid to make time-based decisions in your
business logic          not-rely-on-time
   74:5   warning  Fallback function must be
simple                                   no-complex-fallback
   78:17  warning  Avoid to make time-based decisions in your
business logic          not-rely-on-time
  119:13  warning  Avoid to make time-based decisions in your
business logic          not-rely-on-time
  122:29  warning  Avoid to make time-based decisions in your
business logic          not-rely-on-time
  131:16  warning  Avoid to make time-based decisions in your
```

```
business logic          not-rely-on-time
  185:17  warning  Avoid to make time-based decisions in your
business logic          not-rely-on-time
  215:5   warning  Event and function names must be
different                         no-simple-event-func-name
  273:9   warning  Avoid to use inline assembly. It is
acceptable only in rare cases  no-inline-assembly

✕ 9 problems (0 errors, 9 warnings)
```

Solium output

```
$ solium --file contracts/Auction.sol

contracts/Auction.sol
  59:29     warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  78:16     warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  119:12    warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  122:28    warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  131:15    warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  185:16    warning    Avoid using 'now' (alias to
'block.timestamp').    security/no-block-members
  273:8     error      Avoid using Inline
Assembly.                       security/no-inline-assembly

✕ 1 error, 6 warnings found.
```