SmartDec

BCShop Smart Contracts Security Analysis

This report is public.

Published: February 10, 2018

Abstract

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

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

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 whitepaper; we verify the declared functionality
  - we run tests and check code coverage
- 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 can not be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts. Besides, security audit is not an investment advice.

Checked vulnerabilities

We have scanned BCShop 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

Project Overview

In our analysis we consider:

- BCShop smart contracts code
  - CustomTrancheWallet (commit 0473cef)
  - TokenTrancheWallet (commit 07194fa)
  - TrancheWallet (commit 07194fa)
- tests
  - customTrancheWallet.js (commit 0473cef)
  - trancheWallet.js (commit 20802db)
- description

Project Architecture

The project is a Truffle project.
The files under consideration are two token wallets: TokenTrancheWallet.sol (32 LoC), CustomTrancheWallet.sol (118 LoC).

- CustomTranchWallets
  - Owned.sol (24 LoC)
  - IERC20Token.sol (25 LoC).
- TokenTrancheWallet is inherited from
  - IOwned.sol (7 LoC)
  - TrancheWallet.sol (97 LoC).

The project includes tests.
Compilation of the project fails (however, the contracts that are being audited can be compiled).

Verified Functionality

The wallets store a standard ERC20 token with the following functionality: the owner can define the locking parameters and then lock funds in the contract. After that, in the defined periods of time the defined amount of token will become available for the withdraw by the beneficiary. The owner can change the beneficiary after the funds have been locked, but can not change the parameters of the token locking.

Hereby we verify the following description of the contracts functionality:

“**`CustomTrancheWallet`** contract is intended for token storage and unlocking in accordance with the schedule. The schedule is defined in the constructor and can be changed before the locking function is called with the `setParams` function.
The schedule is set by the two arrays:

- `unlockDates` is an ordered array of unlock dates in the standard Ethereum `timestamp` format (for example 1517443200 is 01 Feb 2018 00:00:00 GMT)
- `unlockAmounts` is an ordered array; each element of `unlockAmounts` is equal to the total amount of tokens available for the withdraw after the corresponding unlock date

For example, if total amount of locked tokens is 100 and we want them to be unlocked in four stages:

- the first stage: 10 tokens unlocked
- the second stage: the rest of the tokens unlocked; 25 tokens available
- the third stage: the rest of the tokens unlocked; 55 tokens available

- the fourth stage: the rest of the tokens (i.e. 45 tokens) unlocked; 100 tokens available

then `unlockAmounts` must be set as [10,25,55,100]. The last element of the array must be equal to the total amount of locked tokens.
Workflow:

1. Create the contract with the desired schedule, stored token and beneficiary.
2. Transfer tokens to the contract.
3. Call `lock` function to lock the tokens.
4. After the tokens are locked, one can check the number of unlocked tokens with `getAvailableAmount` and get all available tokens with `sendToBeneficiary`.

The contract keeps count of unlocked and transferred tokens using `alreadyWithdrawn` field.
After the tokens are transferred to the contract but before the `lock` call it is possible to withdraw all the tokens from the contract.
After the tokens are locked it is possible to change the beneficiary but it is impossible to change lock parameters.

This contract is already deployed to the real network and verified on etherscan.io:

1. Airdrop wallet
ETH address: 0x94Fa60ecD8071597672e937698575aC84bD52b79
Parameters:
On 10 Apr 2018 00:00 GMT 250,000 BCS tokens are unlocked
On 01 Jun 2018 00:00 GMT 1,750,000 BCS tokens are unlocked
The total token number is: 2,000,000

2. Bancor wallet
ETH address: 0xb7804329cab71b5d96cA608A975C7436b23d4DE5
On 10 Apr 2018 00:00 GMT 20,000 BCS tokens are unlocked
On 01 May 2018 00:00 GMT 170,000 BCS tokens are unlocked
The total token number is: 190,000

3. Advisors wallet
ETH address: 0x7F920F9e34573C09B257Ae6e2620aF164158c04e
On 01 May 2018 00:00 GMT 325,000 BCS tokens are unlocked
The total token number is: 325,000

4. Partners wallet
ETH address: 0x2B1Be4591238E0E46401F432a5C6500D804041Bb
On 01 May 2018 00:00 GMT 1,200,000 BCS tokens are unlocked
The total token number is: 1,200,000

5. Long wallet
ETH address: 0xe72017D34F72547793dF611418B228930dA7A7FC
On 01 Jan 2019 00:00 GMT 470,000 BCS tokens are unlocked
On 01 Jan 2020 00:00 GMT 940,000 BCS tokens are unlocked (1,410,000 in total)
On 01 Jan 2021 00:00 GMT 1,410,000 BCS tokens are unlocked (2,820,000 in total)
On 01 Jan 2022 00:00 GMT 1,622,000 BCS tokens are unlocked
The total token number is: 4,442,000

`TokenTrancheWallet` contract is intended for token storage and unlocking in equal portions-tranches. Unlocking periods also have same durations. The initial schedule is set

by the two parameters:

- `tranchePeriodInDays` — time between tranches in days
- `trancheAmountPct` — the size of one tranche in percents of the initial amount of locked tokens

Workflow:

1. Create the contract with the desired tranches parameters, stored token and beneficiary.
2. Transfer tokens to the contract.
3. Call `lock` function to lock the tokens, the number of days for all the tokens to be unlocked should be passed as a parameter. The number of locked tokens is stored in `initialFunds` field.
4. After the tokens are locked, one can check the number of unlocked tokens with `amountAvailableToWithdraw` (which returns the number of available tokens as the first parameter and how many tranches is it as a second parameter) and get all the available tokens with `sendToBeneficiary`.

The contract keeps count of unlocked and transferred tokens using `tranchesSent` field.
After the tokens are transferred to the contract but before the `lock` call it is possible to withdraw all the tokens from the contract.
After the tokens are locked it is possible to change the beneficiary but it is impossible to change lock parameters.

This contract is already deployed to the real network and verified on etherscan.io at the address https://etherscan.io/address/0x700620683311f2d150974b4F262BACFcf7Ff5d8D.
Parameters: 1,500,000 BCS tokens were locked on 11 Jan 2018 11:14:41 GMT for 8 years, every 30 days starting from the first day 1% of the total amount is unlocked (i.e. 1 tranche is available at the time of the audit)”

Automated Analysis

We used several publicly available automated Solidity analysis tools.
Securify does not support 0.4.19 compiler version; the specified version was changed in the code to 0.4.16 for this tool. Securify* showed no issues in the contracts.
Here are the combined results of SmartCheck, Solhint, and Remix.
All the issues found by tools were manually checked (rejected or confirmed).

Tool	Rule	False positive	True positive
SmartCheck	Address Hardcoded		1
	Constant Functions		6
	Dos With Revert	9	
	Erc20 Approve		1
	Erc20 Transfer Should Throw	2	
	Gas Limit And Loops	3	10
	No Payable Fallback		12
	Pragmas Version	1	13
	Reentrancy External Call	13	1
	Unchecked Math	5	3
	Visibility		1
Total SmartCheck		33	48
Remix	Compiler version must be fixed		1
	Explicitly mark visibility of state		1
	Fallback function of contract requires too much gas		1
	Function state mutability can be restricted to pure		1

	Gas requirement of function high	55	11
	Is constant but potentially should not be	2	
	Potential Violation of Checks-Effects-Interaction	5	1
	Potentially should be constant but is not	5	
	Use assert(x) / require(x)	1	
	use of "call"		1
	use of "now"		6
	Variables have very similar names	8	5
Total Remix		76	28
Securify*	Transactions May Affect Ether Receiver	4	
	Transactions May Affects Ether Amount	4	
Total Securify*		8	
Solhint	Avoid to make time-based decisions in your business logic		6
	Code contains empty block		1
	Compiler version must be fixed		14
	Event and function names must be different	1	
	Fallback function must be simple		1
Total Solhint		1	22
Total Overall		**118**	**98**

Securify* — beta version, full version is unavailable.

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

Manual Analysis

Contracts were completely manually analyzed. 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.

Checking balance for equality

The balance is checked for strict equality in CustomTrancheWallet.sol, line 93:

```
require(token.balanceOf(this) == unlockAmounts[unlockAmounts.
    length - 1]);
```

Avoid checking for strict balance equality: an adversary can forcibly send tokens to any account.
We recommend using non-strict inequality.
The issue is known to the developer and is a part of the design.

Reentrancy External Call

There is possible reentrancy in CustomTrancheWallet.sol, line 93:

```
require(token.balanceOf(this) == unlockAmounts[unlockAmounts.
    length - 1]);
```

A malicious external token might execute just `revert();` inside of `balanceOf(this)` and then the wallet will always fall.
The issue is known to the developer and is a part of the design: developer claims that external token contract is audited and can be considered as secure.

Medium severity issues

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

Gas limit and loops

The CustomTrancheWallet contract functions contain loops with unknown number of iterations.

- CustomTrancheWallet.sol, line 58
  `for (uint256 i = unlockDates.length; i != 0; --i)`
- CustomTrancheWallet.sol, line 72
  `for (uint256 i = 0; i < unlockAmounts.length - 1; ++i)`

If the arrays are large enough, the functions will not fit in the block gas limit and the transactions calling it will thus never be confirmed. If the array can be influenced by an attacker (e.g., if an attacker can register arbitrary number of investor accounts), this can lead to an attack.
We highly recommend avoiding loops with big or unknown number of steps.

Timestamp Dependence

`now` is used in

- TrancheWallet.sol, line 64
  `if(now > completeUnlockTime) {`
- TrancheWallet.sol, line 70
  `uint256 periodsSinceLock = (now - lockStart) / (tranchePeriodInDays * 1 days);`
- CustomTrancheWallet.sol, line 52
  `return amountToWithdrawOnDate(now) - alreadyWithdrawn;`
- TrancheWallet.sol, line 43
  `lockStart = now;`

The timestamp of the block can be manipulated by the miner, and therefore should not be used for critical components of the contract. It's quite safe to use `block.timestamp` to change stages. Still, we recommend using `block.number` (and average block time) instead. Other contracts can use `block.number` to stage changes.
Besides, we recommend adding an event for stage change so that it will be more transparent for users.

Unchecked math

Solidity is prone to an integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by malicious account. We recommend using the SafeMath library for all arithmetic operation. It is not used in

- TrancheWallet.sol, lines 44, 70,72, 88

This may lead to overflow.

Low severity issues

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

Pragmas version

Solidity source files indicate the versions of the compiler they can be compiled with.
Example:

```
pragma solidity ^0.4.18; // bad: compiles w 0.4.18 and above
pragma solidity 0.4.18; // good: compiles w 0.4.18 only
```

We recommend following the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. Besides, we recommend using the latest compiler version (0.4.19 at the moment).

Unused variable

The following variable is unused in the contract code

- CustomTrancheWallet.sol, line 20
  `uint256 public initialFunds;`

Dos With Revert

The contract that calls functions of other contracts should not rely on results of these functions.

- CustomTrancheWallet.sol, line 87
  `require(token.transfer(beneficiary, amount));`
- CustomTrancheWallet.sol, line 93

```
        require(token.balanceOf(this) ==
unlockAmounts[unlockAmounts.length - 1]);
```

- TokenTrancheWallet.sol, line 29

```
        require(token.transfer(beneficiary, amount));
```

We recommend being particulary careful when verifying the result of external calls with `require`.

Unchecked math

Solidity is prone to an integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by malicious account. We recommend using the SafeMath library for all arithmetic operation. It is not used in

- CustomTrancheWallet.soll, lines 52, 86
- TrancheWallet.sol, lines 55, 71, 77, 88

This may lead to overflow.

Implicit visibility level

Functions without a visibility modifier are public (i.e., can be called from outside the contract). Make sure this conforms to the intended functionality.

- TrancheWallet.sol, lines 18, 48, 62, 87

Explicitly define function visibility levels (`public`, `private`, `external`, `internal`) to improve code readability.

Conclusion

In this report we have considered the BCShop smart contracts. We performed our analysis according to the procedure described above.
We verify that the functionality of the contracts corresponds to the one described in Verified Functionality section.
The audit showed several security issues of different severity level. We highly recommend addressing them.

This analysis was performed by SmartDec.

COO Sergey Pavlin 

February 10, 2018

Appendix

Code coverage

```
>./node_modules/.bin/solidity-coverage
Running: truffle compile
(this can take a few seconds)...
Error parsing
/home/user/Desktop/mics/bcshop/bcshop.io/coverageEnv/contracts
/misc/ErrorPartnerSale.sol: ParsedContract.sol:125:130:
ParserError: Expected token Comma got 'Semicolon'
(
__StatementCoverageErrorPartnerSale('/home/user/Desktop/mics/b
cshop/bcshop.io/contracts/misc/ErrorPartnerSale.sol',7);
^
Compilation failed. See above.
ls: no such file or directory:
./coverageEnv/build/contracts/*.json
Launched testrpc on port 8555
Running: truffle test
(this can take a few seconds)...
Error parsing
/home/user/Desktop/mics/bcshop/bcshop.io/coverageEnv/contracts
/misc/ErrorPartnerSale.sol: ParsedContract.sol:125:130:
ParserError: Expected token Comma got 'Semicolon'
(
__StatementCoverageErrorPartnerSale('/home/user/Desktop/mics/b
cshop/bcshop.io/contracts/misc/ErrorPartnerSale.sol',7);
^
Compilation failed. See above.
Cleaning up...
Event trace could not be read.
Error: ENOENT: no such file or directory, open
'./allFiredEvents'
Exiting without generating coverage...
```

Compilation output

```
>truffle compile
Error: Spanning multiple lines.
,/home/user/Desktop/mics/bcshop/bcshop.io/contracts/shop/Vendo
r.sol:26:17: DeclarationError: Undeclared identifier.
require(vendorManager.validFactory(msg.sender));
^-----------^
```

```
,/home/user/Desktop/mics/bcshop/bcshop.io/contracts/shop/Vendo
r.sol:45:9: DeclarationError: Undeclared identifier.
vendorManager = manager;
^-----------^
Compilation failed. See above.
```

Tests output

```
>truffle test
Error: Could not find
/home/user/Desktop/mics/bcshop/bcshop.io/test/contracts/helper
s/AddressStorage.sol from any sources; imported from
/home/user/Desktop/mics/bcshop/bcshop.io/test/sol/AddressStora
geUser.sol
at /usr/lib/node_modules/truffle/build/cli.bundled.js:69147:23
at
/usr/lib/node_modules/truffle/build/cli.bundled.js:165637:16
at next
(/usr/lib/node_modules/truffle/build/cli.bundled.js:178009:18)
at /usr/lib/node_modules/truffle/build/cli.bundled.js:69135:7
at /usr/lib/node_modules/truffle/build/cli.bundled.js:202146:5
at
/usr/lib/node_modules/truffle/build/cli.bundled.js:177914:16
at replenish
(/usr/lib/node_modules/truffle/build/cli.bundled.js:165607:25)
at iterateeCallback
(/usr/lib/node_modules/truffle/build/cli.bundled.js:165597:17)
at
/usr/lib/node_modules/truffle/build/cli.bundled.js:165637:16
at ReadFileContext.callback
(/usr/lib/node_modules/truffle/build/cli.bundled.js:202142:14)
```