SmartDec

Brickblock Smart Contracts Security Audit

Abstract

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

Procedure

In our analysis we consider Brickblock whitepaper (version on commit cc4ee61) and smart contracts code (version on commit 76351a4).
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, Oyente 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 run tests and check code coverage
- report
  - we report all the issues found to the developer during the audit process
  - we check the issues fixed by the developer
  - 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.

Checked vulnerabilities

We have scanned Brickblock 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 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

Automated Analysis

We used several publicly available automated Solidity analysis tools.
Securify and Remix do not support 0.4.18 compiler version; the specified version was changed in the code to 0.4.16 for these tools.
Securify and Oyente have not found any bugs.
Here are the combined results of SmartCheck and Remix.
All the issues found by tools were manually checked (rejected or confirmed).

Tool	Vulnerability	false positive	true positive
SmartCheck	Address hardcoded	5	
	Pragmas version		1
	Redundant fallback reject		1
	Reentrancy external call	15	
	Should be pure but is not	7	
	Should be view but is not	8	
	Unchecked math	1	
	Visibility		1
Total SmartCheck		36	3
Remix	Gas requirement of function	6	
	Checks-Effects-Interaction pattern violation	1	1
	Potentially should be constant but is not	1	1
	Inline assembly	1	
	"this" used in constructor	5	
Total Remix		14	2
Overall Total		50	5

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

Code coverage

BrickblockToken.sol is well covered with tests:

- 58/68 of statements (85.29%)
- 23/32 of branches (71.88%)
- 11/12 of functions (91.67%)
- 61/72 of lines (84.72%)

We recommend improving tests for `finalizeTokenSale` function.

Manual Analysis

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

High severity issues

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

The analysis showed no critical issues.

Medium severity issues

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

Contract address

The `upgrade` function (BrickBlockToken.sol, line 147) allows setting the contract's successor, which is supposed to be a contract, too. We highly recommend adding a check that the given address belongs to a contract account and not to an external account.
The issue has been fixed by the developer and is not present on commit 63443af8.

ERC20 concerns

BrickBlockToken contract does not follow some recommendations of ERC20 standard:

> *"A token contract which creates new tokens SHOULD trigger a Transfer event with the `_from` address set to `0x0` when tokens are created."*

We recommend adding `Trasfer` event to constructor.
The issue has been fixed by the developer and is not present on commit 63443af8.

We also recommend modifying `distributeTokens` and `finalizeTokenSale` functions so that they fire `Trasfer` and `Approval` events while working with `balances` and `allowed` mappings.
The issue has been fixed by the developer and is not present on commit 63443af8.

Low severity issues

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

Uninitialized variable

The `fountainContractAddress` variable, which is used at BrickBlockToken.sol, line 114 may be not initialized. We recommend checking this address before using it.
The issue has been fixed by the developer and is not present on commit 63443af8.

No 51% check

The `distributeTokens` function at BrickBlockToken.sol, line 87, allows distributing more than 51% of tokens. This constraint could be checked outside BrickBlockToken contract though. If more than 51% of tokens are distributed then `finalizeTokenSale` function will always throw. We highly recommend adding the check to the `distributeTokens` function.
The issue has been fixed by the developer and is not present on commit 63443af8.

Checks-Effects-Interaction pattern violation

There is a Checks-Effects-Interaction violation at BrickblockToken.sol, line 165. In this case the violations do not lead to actual vulnerabilities.
The issue has been fixed by the developer and is not present on commit 63443af8.

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 (0.4.18 at the moment).
The issue has been fixed by the developer and is not present on commit 63443af8.

Function can be declared as view

In many places in the code there are functions that can be declared as `view`. If a function is not supposed to modify the state, consider declaring it as `view` (alias for `constant`, which will be deprecated for functions). This provides additional checks in case these assumptions are violated. If a function is not supposed to modify the state or read from state, consider declaring it as `pure`.
The issue has been fixed by the developer and is not present on commit 63443af8.

Conclusion

In this report, we have considered the security of Brickblock smart contracts. We performed our audit according to the procedure described above.
The audit showed high code quality and no critical issues. Several medium and low severity issues have been found and reported to the developer. All of them were fixed by the developer and are not present on commit 63443af8.

This analysis was performed by SmartDec

COO Sergey Pavlin 

December 1, 2017