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 BrickblockToken smart contract.

Procedure

In our analysis we consider Brickblock whitepaper (version on commit cc4ee61) and the BrickblockToken smart contract code (version on commit ecebba7).
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, Solhint and Securify (beta version since full version was unavailable at the moment this report was made)
    - we manually verify (reject or confirm) 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
- report
    - we report all the issues found to the developer during the audit process
    - 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 the BrickblockToken smart contract 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 has not found any bugs.
Here are the combined results of SmartCheck, Remix and Oyente.
All the issues found by tools were manually checked (rejected or confirmed).

Tool	Rule	false positives	true positives
SmartCheck	DOS with revert	1	
	Functions returns type and no return		1
	No payable fallback		1
	Reentrancy external call	14	
	Unchecked math	1	
	View function	1	
Total SmartCheck		17	2
Oyente	Source file requires different compiler version	1	
Total Oyente		1	
Remix	Source file requires different compiler version	1	
Total Remix		1	
Solhint	Avoid to use inline assembly	1	
	Not payable fallback	1	
Total Solhint		2	
Overall Total		21	2

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

The contract was completely manually analyzed, its 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.

Critical issues

Critical 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.

Cross-contract interactions

The following issues refer to the behavior of the functions being integrated into other contracts.

1. `toggleDead()` function does nothing for BrickBlockToken contract (no function checks value of dead state variable).
The issue is known to the developer and is a part of design. Developer's comment: the functionality is implemented for the case if the owner of smart contract decides to deploy another replacement contract in the future
2. `distributeTokens(...)` and `distributeBonusTokens(...)` functions are crucial for crowdsale, we haven't checked any contracts using these. They should be called form crowdsale contract only.
The issue is known to the developer and is a part of design. Developer's comment: There is no crowdsale contract in the design. All tokens are being directly distributed from BrickblockToken contract. They are called directly by an owner wallet in cold storage with pre-signed transactions on a cold machine. They are not called by a contract and BBK is not owned by a contract. It is owned by a wallet address.
3. `changeFountainContractAddress` is another important function that must be called in time for any successful crowdsale. We recommend setting `fountainContractAddress` value in constructor if possible.
The issue is known to the developer and is a part of design. Developer's comment: The fountain contract is not yet finalized. This is the reason why the ability to change is implemented in BrickblockToken smart contract. The developer will deploy the fountain contract and change the address before finalizing the token sale.

We have not checked any contracts other than BrickBlockToken.sol, so we cannot say whether these functions are used correctly.

Low severity issues

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

The analysis showed no low severity issues.

Conclusion

In this report we have considered the security of the BrickblockToken smart contract. We performed our audit according to the procedure described above.
The audit showed high code quality and no confirmed issues. However, the scope of work of this audit did not include interactions with other smart contracts. Several cross-contract interactions have been found and reported to the developer as medium issues. All issues were discussed with the developer. Developer's comments can be found in the report.
We highly recommend considering audit of the whole project since some vulnerabilities may arise if the audited contract is used incorrectly by other smart contracts or users.

This analysis was performed by SmartDec

COO Sergey Pavlin [illegible]

January 30, 2018