Token Audit
FO61C77C40C48

INDEX

Overview

This document is a security audit of the contract **PoopWorld.sol** included with this delivery. The token is a standard BEP20 token with a customized logic for transfer functionality.

The scope of this audit is security & vulnerabilities of the token & its adherence to the standard practices which is checked by a standard test. The functionality test of the custom logic is not in the scope.

Following files / results are included with this delivery:

1. Original Contract Code
2. Report (this)
3. Contract Overview diagram (Inheritance & Functionality)

Glossary

The automated tests are performed against a knowledgebase of commonly known issues and assigned a SEVERITY as per the security issue.
Severity is categorized into three levels starting from 1 up to 3. Higher the number, higher is the threat.

- Severity 1 - Low threat
- Severity 2 - Medium threat
- Severity 3 - High threat

Apart from security issues, notes might also be added to point out a certain functionality.

Disclaimer

The audit makes no statements or warrants about the utility of the code, safety of the code, the suitability of the business model, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.

Contract Overview

Note: Contract Diagram has been included with this delivery for tracking contract inheritance, functionality calls.

Contracts:

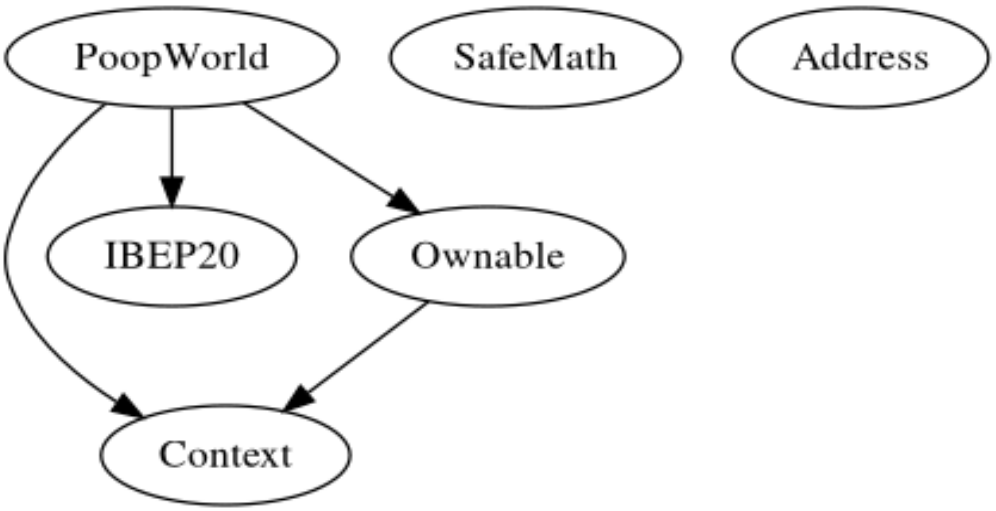
1. **SafeMath** (Library)
    i. Standard SafeMath library has been used for arithmetic operations in this contract to prevent overflows / underflows. This is an industry standard and a safe approach while dealing with numbers in smart contracts.

2. **Context** (abstract contract)
    Provides the “context” for a function call.

3. **IBEP20** (Interface)
    ERC-20 interface

4. **Ownable** (Contract)
    This is a standard ERC20 contract which adds context of an “owner” to the contract which inherits it. Ownership can be transferred and is initially set to the deployer fo the contract

5. **PoopWorld** (Contract, implements *IBEP20,* extends *Ownable, Context*)
    This contract implements IBEP20 and extends the previous contracts.



(Inheritance Diagram)

A detailed map of contract functionality & interdependence can be found in the diagram included with this submission. (**functionality.png**)

Vulnerabilities & Notes - Test Results

This section contains the vulnerabilities & notes which need to be highlighted as per my reasoning mentioned in the description.

Threats

a. Extra Gas In Loops
   i. Severity: 1 (LOW)
   ii. Function / Position:
      1. Lines: 316
      2. Lines: 443
   iii. Description:
   State variable, `.balance`, or `.length` of non-memory array is used in the condition of `for` or `while` loop. In this case, every iteration of the loop consumes extra gas.
   iv. Recommendation:
   If a state variable, `.balance`, or `.length` is used several times, holding its value in a local variable is more gas efficient.

Notes

a. Gas Limit in Loops
   i. Severity: 1 (LOW)
   ii. Function / Position:
      1. Lines: 316
      2. Lines: 443
   iii. Description:
   Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers.
   Moreover, Ethereum miners impose a limit on the total number of gas consumed in a block.
   If `array.length` is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed.
   This limit can be exploited by attackers.

   -> As in this case the ability to manipulate the array named `excluded` is only accessible by the owner, no issue should be caused.
   iv. Mitigation -> (**Not Required**)

(Apart from security audit code has been successfully tested against a standard test suite barring it's custom logistics like transfer())

Final Report

The following are the threats revealed in the security audit of the token contract.

Sr. No.	Threat Name	Type	Files	Severity	Recommendation
1.	Extra Gas In Loops	Extra Gas	PoopWorld.sol	1 (LOW)	Use local variable in loops
2.	Gas Limit in Loops	-	PoopWorld.sol	1 (LOW)	-

Conclusion

This contract Token.sol is a standard ERC20 / BEP20 contract with an added logistics for transfer of tokens. The contract's code is found to be clean and adhering to the ERC20 / BEP20 contracts / interface.

Apart from security audits I have also performed functionality tests of the contract as per ERC20 standard functionality.
The code is functional & the log can be found in this delivery.
Note: Functionalities where a custom logic is used have not been tested due to unavailability of it's logical framework & work-flow to create a test-suite.

A few suggestions can be implemented & have been discussed in depth before.

(Appendix for Threats checked against in the next page).

Appendix

1. Knowledgebase

This audit has tested the contract against the following attacks.

1.1. Unsafe array's length manipulation
1.2. Return value of transfer, transferFrom, or approve function of ERC-20 standard is always false.
1.3. Use of unindexed arguments in ERC-20 standard events
1.4. Costly loop
1.5. Private modifier
1.6. Timestamp dependence
1.7. Use of call function with no data
1.8. Using continue in the do-while loop
1.9. Use of SafeMath
1.10. Blockhash function misuse
1.11. Using tx.origin for authorization
1.12. Standard ERC-20 for functions transfer and transferFrom: return value
1.13. Upgrade code to Solidity 0.5.x
1.14. Implicit visibility level
1.15. Output overwrites input of assembly CALLs
1.16. Non-initialized return value
1.17. Use of assembly
1.18. Use of return in constructor
1.19. Non-strict comparison with zero
1.20. Hardcoded address
1.21. Pure-functions should not read/change state
1.22. Checking for strict balance equality
1.23. Byte array
1.24. constant functions
1.25. Token API violation
1.26. Standard ERC-20 for functions transfer and transferFrom. Exceptions
1.27. DoS by external contract
1.28. Locked money
1.29. Integer division
1.30. Malicious libraries
1.31. Compiler version not fixed
1.32. Private modifier
1.33. Reentrancy
1.34. Specification of the type of function. View-function
1.35. Style guide violation
1.36. Using throw
1.37. Using suicide
1.38. Unchecked math

1.39. Using sha3
1.40. ERC-20 transfer should throw
1.41. Strict comparison with block.timestamp or now
1.42. Overflow and Underflow in Solidity
1.43. Using approve function of the ERC-20 token standard
1.44. Redundant fallback function
1.45. Unsafe type inference
1.46. Revert inside the if-operator
1.47. Private modifier
1.48. Replace multiple return values with a struct
1.49. Specification of the type of function: pure-function
1.50. Non-initialized return value
1.51. Using block.blockhash
1.52. The incompleteness of the compiler: view-function
1.53. Transfer forwards all gas
1.54. No payable fallback function
1.55. DoS by external function call in require
1.56. The incompleteness of the compiler: pure-functions
1.57. Unsafe type inference in the for-loop
1.58. Multiplication after division
1.59. Extra gas consumption
1.60. Costly loop
1.61. Deleting arrays by assigning their length to zero
1.62. Overpowered role
1.63. Send instead of transfer
1.64. msg.value == 0 check
1.65. View-function should not change state
1.66. Unchecked low-level call
1.67. Byte array instead of bytes
1.68. ETH transfer inside the loop
1.69. Deprecated constructions
1.70. Incorrect function signature
1.71. Prefer external to public visibility level
1.72. Deletion of dynamically-sized storage array