StorX Token Audit

INDEX

Overview

This document is a security audit of the contract StorxToken.

Following files / results are included with this audit report:

1. Original Contract Code
2. Report
3. Contract Overview diagram
4. Custom Test Result

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

Contracts:

1. **Initializable** (Contract)

2. **Ownable** (Contract, extends **Initializable**)
   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

3. **Operator** (Contract, extends **Ownable**)

4. **ERC20Basic** (Interface)
   ERC20Basic contract is an interface declaring functionality of ERC20 tokens.

5. **BasicToken** (Interface)
   ERC20 token receiver interface.

6. **ERC20** (Contract, Implements **ERC20Basic**)

7. **StandardToken** (Contract, Implements **ERC20Basic**)

8. **StorxToken** (Contract, extends ERC20 & BasicToken)
   Final contract

A detailed map of contract functionality & interdependence can be found in the diagram included with this report.

Vulnerabilities & Notes - Test Results

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

(*There are no major vulnerabilities found in the StorxToken Contract when checked against the* ***Knowledge base,*** *check Appendix for more*)

(*The Following are the notes which I feel need to be highlighted*)

a. Use of Approve Function - Front-Running Attack
   i. Severity: 2 (MEDIUM)
   ii. Function / Position: approve(); Line 362
   iii. Description:
   The “approve” function of ERC-20 is vulnerable. Using a front-running attack one can spend approved tokens before change of allowance value.
   iv. Attack Possibility Scenario:
   Here is possible attack scenario:
      1. Alice allows Bob to transfer N of Alice's tokens (N>0) by calling approve method on Token smart contract passing Bob's address and N as method arguments
      2. After some time, Alice decides to change from N to M (M>0) the number of Alice's tokens Bob is allowed to transfer, so she calls approve method again, this time passing Bob's address and M as method arguments
      3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls transferFrom method to transfer N Alice's tokens somewhere
      4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain an ability to transfer another M tokens
      5. Before Alice noticed that something went wrong, Bob calls transferFrom method again, this time to transfer M Alice's tokens. In this case max loss can be of M+N tokens
   v. Mitigation:
   A way to mitigate this threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

b. Hardcoded Address
   i. Severity: 1 (LOW)
   ii. Function / Position: Lines: 109; Column: 1
   iii. Description:
   Hardcoded addresses can be used for malicious activities. It needs to be used with caution.

   In this case (StorxToken) it's use is justified.

c. Private Modifier Does Not Hide Data
   i. Severity: 1 (LOW)
   ii. Function / Position: Line 21 / 26 / 66 / 139
   iii. Description:
   Contrary to a popular misconception, the private modifier does not make a variable invisible. Miners have access to all contracts' code and data. Developers must account for the lack of privacy in Ethereum.

   In this case (StorxToken) data is not sensitive, private has been used with the purpose to restrict the variable scope.

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.	Implicit Visibility Level	-	StorxToken.sol	1 (LOW)	Explicitly declare visibility
2.	Use of Approve Function - Front-Running Attack	-	StorxToken.sol	2 (MEDIUM)	Mitigation mentioned
3.	Private Modifier Does Not Hide data	-	StorxToken.sol	-	-

Conclusion

This contract StorxToken.sol is a standard ERC20 contract with an added functionality for burn, mint and operable. The contract's code is found to be clean and adhering to the ERC20 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 along with the report.

A few points have been highlighted as per my thinking.

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