BRIDGE TOKEN AUDIT REPORT

TRC20 Token

XinFin Fintech Pte Ltd

August 21, 2020

Overview

This document is a security audit of Bridge Token's token contract by XinFin Team. The scope of the security audit was restricted to:

a. Scan the contracts listed above for generic security issues using automated systems and manually inspecting the results.
b. Manually auditing the contracts listed above for security issues.

The following are contracts/files audited by the team from the Github repository:
https://github.com/cryptoland-blockchain-laboratory/Bridge-token/

a. BasicToken.sol
b. BridgeToken.sol
c. Ownable.sol
d. Pauseable.sol
e. SafeMath.sol
f. SmartToken.sol
g. StandardToken.sol

Automated tests were performed on both Source Code and compiled bytecode of the above smart contracts.

Glossary

The automated tests are performed against a knowledgebase of commonly known issues and assigned a SEVERITY as per the security issue. You can find the list of threats in the knowledgebase at the end in the Appendix section.

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. Notes are not security issues or threats but only points which we feel need to highlight.

Vulnerabilities

The vulnerabilities are listed below as per the file in which they occur along with their severity and suggested changes.

1. Common - in all Smart Contracts
    1.1. Solidity compiler version
        1.1.1. Severity: 1 (LOW)
        1.1.2. Function / Position: First line of each contract
        1.1.3. Description:
        It is recommended to use a fixed version of the compiler, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

```
pragma solidity ^0.5.8;
```

        1.1.4. Recommendation:
        Specify the exact compiler version (pragma solidity 0.5.8).

```
pragma solidity 0.5.8;
```

Notes

Notes are not security issues or threats but only points which we feel need to highlight.

1. Use of Private Modifiers
    1.1. 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.
    1.2. Type: Privacy
    1.3. Positions:
        1.3.1. Contract Name: Pauseable.sol
            1.3.1.1. Line: 29; Column: 9
        1.3.2. Contract Name: StandardToken.sol
            1.3.2.1. Line: 24; Column: 52
        1.3.3. Contract Name: Ownable.sol
            1.3.3.1. Line: 61; Column: 8
        1.3.4. Contract Name: BridgeToken.sol
            1.3.4.1. Line: 6; Column: 11
            1.3.4.2. Line: 7; Column: 11
            1.3.4.3. Line: 8; Column: 10
2. External Call To User-Supplied Address
    2.1. Description: An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.
    2.2. Function Name (external call): transferAndCall(address,uint256,bytes)
    2.3. Type: Security
    2.4. Position: SmartToken.sol:49

```
receiver.onTokenTransfer(msg.sender, _value, _data)
```

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	Non-Fixed compiler version	-	All Contracts	1 (LOW)	Use a fixed compiler version

Notes to be considered.

Sr. No.	Note Name	Description	Type	Files
1.	Use of private modifiers	'Private' does not hide data	Privacy	Pauseable.sol, StandardToken.sol, BridgeToken.sol, Ownable.sol
2.	External Address Call	Ensure reentrancy guards are in place	Security	SmartToken.sol

Conclusion

The contract implements the TRC-20 standard functions, events and state variables, and explicitly defines the visibility of each function.

Overall we found the contract code to be clean and thoughtfully written with no major security issues or threats. We recommend a few minor changes & notes to keep in mind which have been mentioned in depth in the previous sections.

Appendix

a. Knowledgebase

The following are the 72 attacks which were tested for in our automated and manual testing environment:

1. Unsafe array's length manipulation
2. Return value of transfer, transferFrom, or approve function of ERC-20 standard is always false.
3. Use of unindexed arguments in ERC-20 standard events
4. Costly loop
5. Private modifier
6. Timestamp dependence
7. Use of call function with no data
8. Using continue in the do-while loop
9. Use of SafeMath
10. Blockhash function misuse
11. Using tx.origin for authorization
12. Standard ERC-20 for functions transfer and transferFrom: return value
13. Upgrade code to Solidity 0.5.x
14. Implicit visibility level
15. Output overwrites input of assembly CALLs
16. Non-initialized return value
17. Use of assembly
18. Use of return in constructor
19. Non-strict comparison with zero
20. Hardcoded address
21. Pure-functions should not read/change state
22. Checking for strict balance equality
23. Byte array
24. constant functions
25. Token API violation
26. Standard ERC-20 for functions transfer and transferFrom. Exceptions
27. DoS by external contract
28. Locked money
29. Integer division
30. Malicious libraries
31. Compiler version not fixed
32. Private modifier
33. Reentrancy
34. Specification of the type of function. View-function

35. Style guide violation
36. Using throw
37. Using suicide
38. Unchecked math
39. Using sha3
40. ERC-20 transfer should throw
41. Strict comparison with block.timestamp or now
42. Overflow and Underflow in Solidity
43. Using approve function of the ERC-20 token standard
44. Redundant fallback function
45. Unsafe type inference
46. Revert inside the if-operator
47. Private modifier
48. Replace multiple return values with a struct
49. Specification of the type of function: pure-function
50. Non-initialized return value
51. Using block.blockhash
52. The incompleteness of the compiler: view-function
53. Transfer forwards all gas
54. No payable fallback function
55. DoS by external function call in require
56. The incompleteness of the compiler: pure-functions
57. Unsafe type inference in the for-loop
58. Multiplication after division
59. Extra gas consumption
60. Costly loop
61. Deleting arrays by assigning their length to zero
62. Overpowered role
63. Send instead of transfer
64. msg.value == 0 check
65. View-function should not change state
66. Unchecked low-level call
67. Byte array instead of bytes
68. ETH transfer inside the loop
69. Deprecated constructions
70. Incorrect function signature
71. Prefer external to public visibility level
72. Deletion of dynamically-sized storage array

Disclaimer

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