Health Fabric
Smart Contract Audit Report

July 08, 2021

IMMUNE BYTES

Introduction

1. About Health Fabric

Health Fabric has been a proven platform for the management of long-term conditions, and is releasing a new version of their product called 'Unity' which encompasses a new addition of blockchain technologies and the Health Fabric token that will transform the way healthcare is delivered and enable a new experience for service users to access trusted verified, self management content, accessible across different communities, and allow service users to share their health activities, outcomes and monetise their data, whilst helping others.

The blockchain integration enables clinicians to create traceable, self help content, published to the blockchain accessed by service users with multiple long-term conditions. This compensates and empowers clinicians to create self-management content in a trusted environment.

Visit https://www.healthfabric.co.uk/ to learn more.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The Health Fabric team has provided the following doc for the purpose of audit:

1. HFabric_TokenSpecification v1.0.docx(2).pdf
2. HealthFabricWhitePaper1.2.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Health Fabric
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: 0deefb2659dc9dae29fac2c18a33933ee3176847
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	No issues found	No issues found	No issues found
Closed			

High severity issues

No issues found.

Medium severity issues

No issues found.

Low severity issues

No issues found.

Unit Test

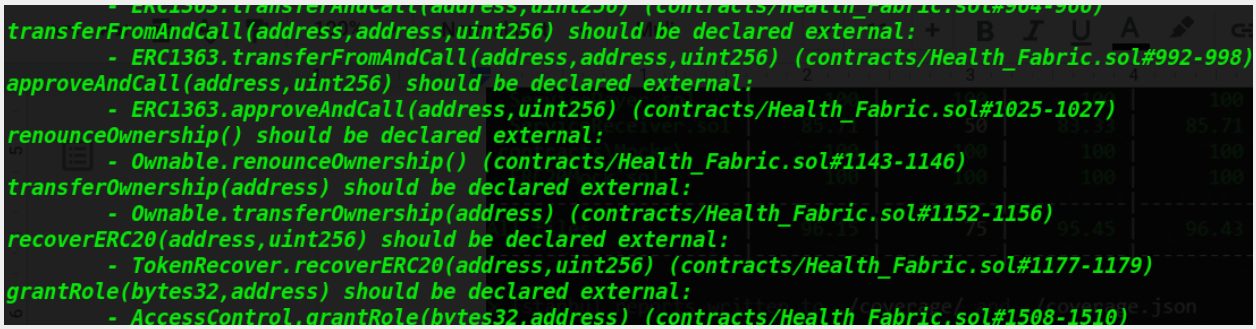
```
Contract: PowerfullERC20
  contract deployment
    √ supports erc1363 interface (80ms)
    √ should have max cap of 1b (823ms)
    √ owner initial balance & token decimals (114ms)
    √ service fee is set & can be withdrawn by owner (383ms)
    √ owner has minter role & can grant others (567ms)
    √ user can mint tokens (318ms)
    √ should recover lost erc20 tokens (818ms)
    √ cannot mint after finishMinting is called (641ms)


  8 passing (4s)
```

Coverage Report

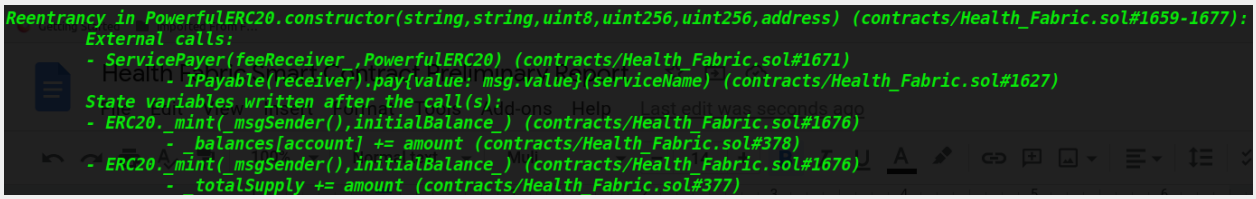
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	96	75	95.24	96.3	
ERC20Decimals.sol	100	100	100	100	
ERC20Mintable.sol	100	100	100	100	
HFABRIC.sol	100	50	100	100	
Roles.sol	100	100	100	100	
ServicePayer.sol	100	100	100	100	
ServiceReceiver.sol	85.71	50	83.33	85.71	34
contracts\Mocks\	100	100	100	100	
ERC20Mock.sol	100	100	100	100	
All files	96.15	75	95.45	96.43	

```
> Istanbul reports written to ./coverage/ and ./coverage.json
> solidity-coverage cleaning up, shutting down ganache server
```

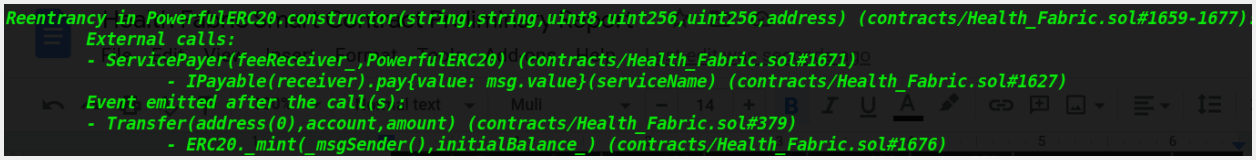
Automated Audit



```
transferFromAndCall(address,address,uint256) should be declared external:
        - ERC1363.transferFromAndCall(address,address,uint256) (contracts/Health_Fabric.sol#992-998)
approveAndCall(address,uint256) should be declared external:
        - ERC1363.approveAndCall(address,uint256) (contracts/Health_Fabric.sol#1025-1027)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/Health_Fabric.sol#1143-1146)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/Health_Fabric.sol#1152-1156)
recoverERC20(address,uint256) should be declared external:
        - TokenRecover.recoverERC20(address,uint256) (contracts/Health_Fabric.sol#1177-1179)
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (contracts/Health_Fabric.sol#1508-1510)
```



```
Reentrancy in PowerfulERC20.constructor(string,string,uint8,uint256,uint256,address) (contracts/Health_Fabric.sol#1659-1677):
        External calls:
        - ServicePayer(feeReceiver_,PowerfulERC20) (contracts/Health_Fabric.sol#1671)
                - IPayable(receiver).pay{value: msg.value}(serviceName) (contracts/Health_Fabric.sol#1627)
        State variables written after the call(s):
        - ERC20._mint(_msgSender(),initialBalance_) (contracts/Health_Fabric.sol#1676)
                - _balances[account] += amount (contracts/Health_Fabric.sol#378)
        - ERC20._mint(_msgSender(),initialBalance_) (contracts/Health_Fabric.sol#1676)
                - _totalSupply += amount (contracts/Health_Fabric.sol#377)
```



```
Reentrancy in PowerfulERC20.constructor(string,string,uint8,uint256,uint256,address) (contracts/Health_Fabric.sol#1659-1677):
        External calls:
        - ServicePayer(feeReceiver_,PowerfulERC20) (contracts/Health_Fabric.sol#1671)
                - IPayable(receiver).pay{value: msg.value}(serviceName) (contracts/Health_Fabric.sol#1627)
        Event emitted after the call(s):
        - Transfer(address(0),account,amount) (contracts/Health_Fabric.sol#379)
                - ERC20._mint(_msgSender(),initialBalance_) (contracts/Health_Fabric.sol#1676)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Concluding Remarks

While conducting the audits of the Health Fabric smart contract, it was observed that the contract does not contain any High, Medium, and Low severity issues.

Notes

- ***The Health Fabric team has followed best practices for Smart Contract development and implemented standard ERC20.***

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Health Fabric platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.