

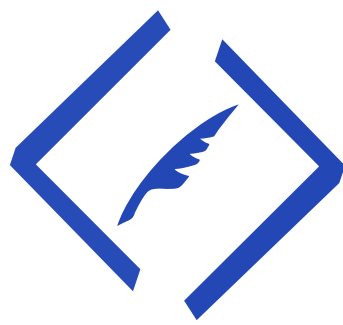
Digipharma

Smart Contract Audit Report

[View Project](#) >

Prepared on April 21, 2019





QuillAudits

Redefining **Blockchain** Security Standards

QuillAudits offers advanced Ethereum, EOS, TRON smart contract audit, blockchain protocol security, and formal verification to ensure your platform's integrity.

Index

01	Introduction
02	Approach & Methodologies
03	Audit Details & Goals
04	Security Level References
05	High Severity Issues
06	Medium Severity Issues
07	Low Severity Issues
08	Unit Testing
12	Code Coverage
13	Manual Testing
14	Recommendations

Introduction

This Audit Report highlights the overall security of **Digipharm** Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

DIGIPHARMS' solutions aim to accelerate the much needed restructuring of healthcare delivery systems to a value-based approach. The inherent flaws of current global healthcare infrastructure result in substantial inefficiency, wastage and sub-optimal health outcomes for the most important stakeholder; the patient.

In healthcare today, there is a lack of incentive for health systems and providers to focus on mutually beneficial collaboration and the maximisation of patient outcomes.

DIGIPHARM utilises blockchain technology to overcome key infrastructural limitations to innovative pricing and evidence generation. DIGIPHARMS' platforms drive value improvement, reward innovation and expedite the transformation to value-based, personalised healthcare.

Visit Website >

Approach & Methodologies

This Audit Report highlights the overall security of Digipharm Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Approach

QuillAudits team has performed thorough testing of the project by beginning 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 race conditions, transaction-ordering dependence, timestamp dependence and denial of service attacks.

In the Unit testing phase we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

Collaboration among QuillAudits Team

The code was tested in collaboration of our multiple team members and it included:

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

Audit Details & Goals

Project Name: DIGIPHARM

Website: <https://www.digipharm.io>

Languages: Solidity (Smart Contract), Javascript(Unit Testing)

Contract Summary

QuillAudits conducted a security audit of a smart contract of digipharm. Digipharm contract is used to create the ERC20 token which is a DIGIPHARM TOKEN, Smart contract contain basic functionalities of ERC20 token with total supply of 1 billion and some advance functionalities.

- Stop transfer functionality (by owner of contract)
- Start/Resume transfer functionality (by owner of contract)
- Lock transfer functionality for particular user, that required amount of tokens to be lock and time to release tokens that are locked.

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 in the following three categories:

Security: Identifying security related issues within each contract and within the system of contracts.




Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security Level References

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


No.	Issue	Label
01	High Severity Issues that will probably bring problems and should be fixed.	
02	Medium Severity Issues that could potentially bring problems and should eventually be fixed.	
03	Low Severity Issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.	

No.	Status	Label
01	Issues Fixed by Developer	
02	Issues not Fixed	

No. of issues per severity

	Low	Medium	High
Open	0	0	0
Closed	4	0	1

High Severity Issues

No.	Issue	Status
01	<p>canBeTransferred() function of Digipharmtoken.sol contract should be converted to a view function because this function is only responsible for returning boolean value and doesn't alter storage state in any way.</p> <p>When you don't use view modifier then that function is considered as setter function which means that function is intended to update the storage of smart contract and to call that function Gas will be required, on the other hand if you use view functionality that will consider function as getter function which means that function is only reading from smart contract storage and doesn't required gas to call function. This function is only reading from smart contract storage so it must be converted to view function.</p> <div><div>canBeTransferred</div><div>"0xca35b7d915458ef540ade6068dfe2f44e8fa733c", "100"</div><div>▼</div></div> <div><div>canBeTransferred</div><div>address addr, uint256 value</div><div>▼</div></div> <p>As you can see in above pictures, picture 1 depicts that when you don't use view modifier then that function is considered as a setter function also in return you will not get the return result, but when view modifier is used function will be converted into getter function and no transaction cost is required to get data.</p>	







Medium Severity Issues

No Medium Severity Issues are present.

Medium Severity Issues




















Low Severity Issues

No.	Issue	Status
01	<p>Solidity version must be fixed (Always use latest Version).</p> <p>It should not be <code>pragma solidity ^0.4.24;</code> It should be <code>pragma solidity 0.4.24;</code></p> <p>Version should be fixed so that development phase and deployment phase should have same solidity version.</p>	
02	<p>Constructor of Contract ownable.sol is declared as of its contracts name - <code>function ownable()</code>, declaring constructor as a same name of function is deprecated use <code>constructor()</code>.</p>	
03	<p>Invoking event without “<code>emit</code>” prefix is deprecated use <code>emit</code> while invoking events at all places.</p> <p>Example:</p> <p>Contract : MintableToken.sol</p> <p>Event: <code>Mint()</code>, <code>Transfer()</code>, <code>MintFinished()</code>.</p>	
04	<p>Always use access modifier <code>public/private/external/internal</code> while declaring functions, In DigipharmToken.sol Contract six functions doesn't contain access modifiers.</p> <p>Functions: <code>burn()</code>, <code>burnFrom()</code>, <code>canBeTransferred()</code>, <code>transfer()</code>, <code>stopTransfer()</code>, <code>resumeTransfer()</code>.</p> <p><code>burn(uint256 _value)</code> <code>public</code> onlyOwner <code>burnFrom(uint257 _value, address victim)</code> <code>public</code> onlyOwner <code>canBeTransferred(address victim, uint256 _value)</code> <code>public</code> view <code>transfer(address _to, uint256 _value)</code> <code>public</code> <code>stopTransfer()</code> <code>public</code> onlyOwner <code>resumeTransfer()</code> <code>public</code> onlyOwner</p>	


Unit Testing

Following test cases were used for Unit Testing

No.	Test Case	Result
01	Should correctly initialize constructor values of Digipharm Token Contract (174ms)	
02	Should check the Total Supply of DIGIPHARM TOKEN (44ms)	
03	Should check the Maximum Total Supply of DIGIPHARM TOKEN (56ms)	
04	Should check the Name of a token of DIGIPHARM TOKEN (60ms)	
05	Should check the symbol of a token of DIGIPHARM TOKEN (46ms)	
06	Should check the decimal of a token of DIGIPHARM TOKEN (52ms)	
07	Should check the Owner of a DIGIPHARM TOKEN contract (39ms)	
08	Should check the balance of a Owner (57ms)	
09	Should check transfered allowed or Not (46ms)	
10	Should not be able to Mint token by Non Owner account (169ms)	
11	Should Mint token by Owner to Account [1] (302ms)	
12	Should check the balance of a Account [1] after minting	
13	Should check the Total Supply of DIGIPHARM TOKEN after Minting Token (40ms)	
14	Should not be able to burn tokens of Account [1] more than account balance (91ms)	
15	Should not be able to burn tokens of Account [1] by Non owner Account (111ms)	
16	Should be able to burn tokens of Account [1] by owner only (136ms)	
17	Should check the balance of a Account [1] after tokens burned by Owner (92ms)	

Unit Testing

Following test cases were used for Unit Testing

No.	Test Case	Result
18	Should check the Total Supply of DIGIPHARM TOKEN after tokens are burned	
19	Should be able to Mint tokens by Owner to Owner (362ms)	
20	Should check the balance of a Account [0] after minting	
21	Should check the Total Supply of DIGIPHARM TOKEN after Minting Token	
22	Owner Should be able to burn tokens using function Burn (225ms)	
23	Should check the balance of Owner after burn	
24	Should check the Total Supply of DIGIPHARM TOKEN after Burn	
25	Should be able to transfer tokens to accounts[2] by owner (195ms)	
26	Should Not be able to transfer tokens to accounts[3] by accounts[2] when transfer is Not allowed (188ms)	
27	Should be able to Mint tokens till maximum allowed (184ms)	
28	Should Not be able to Mint tokens more than maxm Supply (155ms)	
29	Should Not Resume transferred allowed by Non owner Account (84ms)	
30	Should Resume transferred allowed (123ms)	
31	Should be able to transfer tokens to accounts[5] by non Owner account when transfer allowed (226ms)	
32	Should check minting finished or not when total supply is equal to maxm supply (38ms)	
33	Should Not update state of minting finish by Non owner Account (75ms)	

Unit Testing

Following test cases were used for Unit Testing

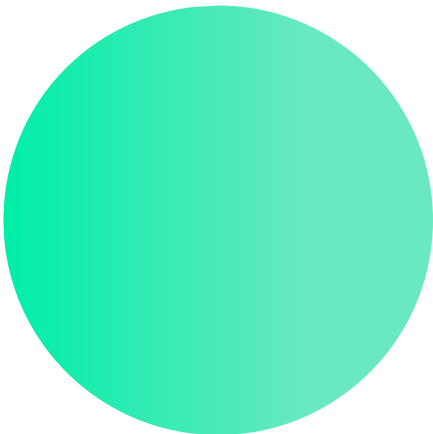
No.	Test Case	Result
34	Should update state of minting finish (115ms)	
35	Should Not Stop transfer functionality by Non Owner Account (66ms)	
36	Should Stop transfer functionality by owner Account (118ms)	
37	Should Not be able to transfer tokens to accounts[3] by accounts[2] when transfer is Not allowed (161ms)	
38	Should Not Resume transfer functionality by Non Account (77ms)	
39	Should Resume transfer functionality by owner Account (129ms)	
40	Should Approve address[5] to spend specific token on the behalf of accounts[3] (83ms)	
41	Should Not be able to transfer tokens more then allowance figure (127ms)	
42	Should be able to transfer tokens to accounts[3] it self after approval from accounts[2] (211ms)	
43	Should check the allowance after transferring tokens on the behalf of accounts[2]	 
44	Should increase the allowance (133ms)	
45	Should decrease the allowance (147ms)	
46	Should Not be able to transfer ownership of Digipharma token Contract from Non Owner Account (84ms)	
47	Should be able to transfer ownership of Digipharma token Contract (135ms)	
48	Should be able to check if user can transfer tokens or not	
49	Should be able to lock tokens of accounts[3] for particular period of time (116ms)	

Unit Testing

Following test cases were used for Unit Testing

No.	Test Case	Result
50	Should Not be able to transfer tokens to accounts[2] by accounts[3] when lock (538ms)	✓
51	Should be able to transfer tokens to accounts[2] by accounts[3] when locked but value less then locked value (392ms)	✓
52	Should transfer tokens to account[3] to check token transfer after lock period (151ms)	✓
53	Should be able to transfer tokens to accounts[2] by accounts[3] when locked Period is over (334ms)	✓

Result



✓ 53 Passed

✗ 00 Failed

Code Coverage

Coverage Report

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	90.36	63.64	92.59	90.8	
BasicToken.sol	100	50	100	100	
DigipharmToken.sol	100	83.33	100	100	
ERC20.sol	100	100	100	100	
ERC20Basic.sol	100	100	100	100	
MintableToken.sol	100	50	100	100	
Ownable.sol	100	75	100	100	
SafeMath.sol	41.67	25	50	41.67	... 18,19,27,29
StandardToken.sol	95.24	62.5	100	95.24	87
All files	90.36	63.64	92.59	90.8	

Slither Tool Result

```

Different versions of Solidity is used in :
- Version used: ['>=0.4.21<0.6.0', '^0.4.18', '^0.4.24']
- digipharm/contracts/MintableToken.sol#1 declares pragma solidity^0.4.18
- digipharm/contracts/StandardToken.sol#1 declares pragma solidity^0.4.18
- digipharm/contracts/DigipharmToken.sol#1 declares pragma solidity^0.4.24
- digipharm/contracts/ERC20.sol#1 declares pragma solidity^0.4.18
- digipharm/contracts/ERC20Basic.sol#1 declares pragma solidity^0.4.18
- digipharm/contracts/Migrations.sol#1 declares pragma solidity>=0.4.21<0.6.0
- digipharm/contracts/Ownable.sol#1 declares pragma solidity^0.4.18
- digipharm/contracts/BasicToken.sol#1 declares pragma solidity^0.4.18
- digipharm/contracts/SafeMath.sol#1 declares pragma solidity^0.4.18

```

File Description Table

File Name	SHA-1 Hash
BasicToken.sol	753c08ed304a39a35fc697ab5af8d442d4243a3e
DigipharmToken.sol	78b03e3c3c19d53ea9593ea0cc5611a35bdb8a23
ERC20Basic.sol	c18b2a506949adb6991788c4269b1ade2885e9d9
ERC20.sol	fad5ec53b37c11c9c9d38491a5badf27695e8684
Migrations.sol	77c7f03cd8da7de52e322270d97f0b1952fcaef6
MintableToken.sol	4fee422050e6bc8dd6df9bed0d1dbef4189c3fb7
Ownable.sol	90b9233fbde1bdd37277e16b67267cae81ae8ddb
SafeMath.sol	8158287bb2237f789a99d3eb7b565f8334f00493
StandardToken.sol	72628364ac43d12576681b0e4be5dfcc880d4f0f

Manual Testing

Network: Rinkeby

No.	Transaction Hash
01	Contract Deployed: 0xdf29ca7c400de81dacdc29a36f8a9264fa6e58d12c299f9b9bd210db6bae41d7 Digipharm Contract Address: 0xe0e1c1ef48104499ed7308e412d0c5d03ede70ad
02	0xaf12de52cfd5ca67f76db2a6391c1c17cd02c1baaa2537f3ba997a9e5e7de34a Minted 100 DPH Tokens by owner
03	0x19dc622d1ff088f9d202f1e473c87256192e99de6f872b4990b041f348f40434 Minted 99999900 and completed total supply
04	0x28f3337fddd969fb3fab4c4238777c048723b6d0d9f628a3ad618cd555ee94ae failed : transfer tokend from non owner account when transfer is not allowed - Passed
05	0x109adacf657c53fab5d6cb7df4fdfe5e8770d6b72c83a068b295ae7bde38b93b minting finished
06	0x4bec67469e045659cb8e2bb417220704378768d543f02035f69b6834f8fca04c transaction failed when try to mint tokens after minting is finished
07	0x0532c5ee5e931d6f73780358fef234179e641ff325cfb4214376cc3c290d99f2 Resume transfer
08	0xa271031bea0e16bed60b5105cd3a169d71e501672b538331e533771c9fa109bd transfer : after transfer is allowed (from non owner account)
09	0x86ba75ac66d973a8e737ad4439a99ee4abb9ce5b889ad717b09967b1ba80b89e burn from failed by Nin owner account.
10	0x5833b0243be17aa8e00a88a64f2dadcaf8f1c8f25ec8a2eb735b09b26d37af4c failed - should not be able to burn when minting is finished.
11	0xd5b160690cf66e4ed060aa49c02e28b474865f8ecfabcb27a1378e3f84279935f stop transfer only by owner

Manual Testing

Network: Rinkeby

No.	Transaction Hash
12	<code>0xbe4438774772e18a520070e21cffd824f91676f129a2e6e68a865b1f3229be29</code> burn tokens of owner
13	<code>0x225711a7a229a2e7ed8a4dd4f89843f1209baa5df85bffafcb7f406c30de0d7f</code> lock tokens by owner
14	<code>0xe05f6546f5bed7114d2e598f7442da563063cfbec072b2beca2be42300306379</code> trying to transfer token more then locked by owner (should fail)
15	<code>0x9867d89a334b95136d14e33df70a732a09d2951b3eb8f11a46432a37014d7eb4</code> trying to transfer tokens less then locked
16	<code>0x0b488e5ad55134b963b195f3fdd10354509a69c2288114e2196a7cf3f525433a</code> can be transfered function call (Failed to get data in response true/false)
17	<code>0x0f79da41549e35cfa4f9fb33beb0f8e9e4cfbb3e8c28e674e795e825e3dfb665</code> Transfer success after transfer lock period is over.
18	<code>0x4ec3dd824adeebbbb1240e37db1b5e3b98596043a0f37a833fa161f4af17cd44</code> transfer ownership call by non owner account
19	<code>0x69ba5ae1fc3978a999b8b0a546c3cf1d19c7c331d2ab551a051fc0126dad0dca</code> transfer ownership by owner
20	<code>0x5dc3a51acf4c3997c779a9c71c1c26b3d8b230ca2abaa5bc79afb741ceae93a5</code> approves function called
21	<code>0x276641de3bb93d71d073e65e3eea5cf33b0e12c7b03986a5969ee4e2db12542d</code> transfer from should fail (traying to transfer tokens more then approved)
22	<code>0xf33ea04099508eac5af016c6662dc50b37d4538e94166e3984b126fe4fc6f537</code> tranfer from function called

Recommendations

Implementations

All the implementations and recommendations of Initial audit report is accepted by Digipharm development team.

Comments

Use case of smart contract is very well designed and Implemented. Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity. DIGIPHARM development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation .

All the High, Medium, low level issues are resolved by digipharm development team.



Redefining **Blockchain** Security Standards

Write to us at: hello@quillhash.com

Visit us at: <https://audits.quillhash.com/>

Follow us on

