McAfeeDex Security Audit

McAfeeDex security audit, conducted by Callisto Network Security Department in October 2019.

McAfeeDex Specificities

Audit Request

McAfeeDex is intended to serve as a truly decentralized exchange with multiple UI implementations provided by different parties independently.

Source Code:

- https://github.com/Mshuu/MCAFEEDEXCONTRACT.

Disclosure policy:

Standard disclosure policy.

Platform:

ETH.

McAfeeDex Security Audit Report

Are Your Funds Safe?

1. In scope

Commit hash 5db8389ce23ce11fb273c9573772287aa75ce43b.

- contract.sol.

2. Findings

In total, **20 issues** were reported including:

- 1 high severity issue.
- 3 medium severity issues.
- 6 low severity issues.
- 7 notes.
- 3 owner privileges (the ability of an owner to manipulate contract).

No critical security issues were found.

`StandardToken` and `ReserveToken` contracts issues

2.1. Known vulnerabilities of ERC-20 token

Severity: low.

Description:

1. Double withdrawal attack is possible. More details here.
2. Lack of transaction handling mechanism. WARNING! This is a very common issue, and it already caused millions of dollars losses for lots of token users! More details here.

Recommendation:

Add the following code to the `transfer(_to address, ...)` function:

```
require( _to != address(this) );
```

2.2. ERC20 Compliance: false instead of throw

Severity: medium.

Description:
From ERC-20 specification:

```
    The function SHOULD throw if the _from account balance does not have enough tokens to spend.
```

In the implementation of McAfeeDEX a function returns `false` instead. This can lead to serious consequences for 3d party developers who work with this contract.

For example, an external contract may use this token contract as:

```
AdChainToken.transferFrom(recipient, this, value);
points[recipient] += value;
```

In this case, the recipient will get the increase of points, but the transfer of tokens will not happen. According to the definition of ERC20 standard the transaction must be interrupted by the call of `throw` at the token contract, however in case of the audited smart-contract the code will successfully complete the transaction.

Code snippet:

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L67-L90

2.3. ERC20 Compliance

Severity: low.

Description:

According to the ERC20 standard, a transfer event must be generated when the token contract is initialized, if any token value is set to any given address.

Code snippet:

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L120

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L126

2.4. Token Transfer to 0x0 address

Severity: low.

Description:

Transfer & transferFrom functions do not prevent from sending tokens to address 0x0.

Code snippet:

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L67

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L80

Recommendation:

Add zero address checking.

```
require(to != address(0));
```

2.5. Known vulnerabilities of ERC-20 token

Severity: medium.

Description:

Both `transfer` and `transferFrom` can not process transfers of 0 tokens.

The condition `balances[_to] + _value > balances[_to]` means that `value` cannot be equal to zero.

Please note that "Transfers of 0 values MUST be treated as normal transfers and fire the Transfer event" following ERC20 standard.
This issue can create compatibility issues with contracts that rely on ERC20 standard definition.

Please refer to the ~~rfc~~ definition of "MUST" and "SHOULD" to correctly implement ERC20.

Code snippet:

https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L67

https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L80

`SwitchDex` contract issues

2.6. Lack of ERC20 extraction functions

Severity: high.

Description:

Warning! This problem will lead to huge financial losses for McAfeeDEX customers.

There is a critical flaw in the design of ERC20 token standard. ERC20 tokens lack event emitting/handling.

This makes it impossible for the recipient contract to reject incorrect transactions of ERC20 tokens. For the McAfeeDEX contract, it means that users can (and will) deposit their tokens via the `transfer` function, and the transaction will be completed successfully. Tokens will not be credited to the customer's account. These tokens will be trapped inside the contract forever without the possibility of their recovery.

Knowing this, it is necessary to implement a function that will allow to withdraw "trapped" tokens from the contract and send them back to users.

Recommendation:

It is necessary to track the amount of tokens that were deposited properly. In this case, the developer can compare the amount of tokens that the McAfeeDEX contract owns to the amount of tokens that were recorded as "properly deposited". Then the developer can extract the excess of tokens to send them back to users manually.

```
contract SwitchDex is SafeMath {

  mapping (address => uint256) public total_deposited;

  ...

    function depositToken(address token, uint amount) {
    //remember to call Token(address).approve(this, amount) or this contract will not be able to do the transfer on your behalf.
    if (token==0) throw;
    if (!Token(token).transferFrom(msg.sender, this, amount)) throw;
    tokens[token][msg.sender] = safeAdd(tokens[token][msg.sender], amount);
    Deposit(token, msg.sender, amount, tokens[token][msg.sender]);

    // Record tokens as properly deposited
    total_deposited[token] = safeAdd( total_deposited[token], amount );
  }

  function extractERC20(address _token) {
    if (msg.sender != admin) throw;

    uint256 actual_balance = StandardToken( _token ).balanceOf( this );
    StandardToken( _token ).transfer( msg.sender, safeSub( actual_balance, total_deposited[_token] ) );
  }
}
```

You could also use an alternative smart-contract development platform that is not prone to these problems to help your users.

2.7. Test Trade

Severity: medium.

Description:

When `testTrade` is used the `tokens[tokenGet][sender]` is checked to be higher or equal to `amount`, which is wrong since taker fees are also substructed from `msg.sender` token balance or ether balance. This means that in order for the test to be more correct, it is necessary to calculate the fee, add it to the amount and check against `tokens[tokenGet][sender]`.

This issue will lead to the fact that the transaction of a user who wants to exchange the entire balance of one asset for another will be thrown if the amount is equal to their total balance, since they will not have enough tokens to cover the exchange cost.

Please note that this issue is applicable for users with `accountLevels` set to false.

The `testTrade` function is used in the UI to prevent users from making incorrect trades. However, as explained above, this can allow users to accept the offer and fail to execute it, causing them to lose all transaction gas since `revert` is not used.

Code snippet:

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L278#L284~~

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L271~~

2.8. No checking for zero address

Severity: low.

Description:

Incoming addresses should be checked for an empty value(0x0 address) to avoid loss of funds or to block some functionality.

Code snippet:

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L180

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L190

https://github.com/Mshuu/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L185

2.9. Dex Admin Update Mechanism

Severity: low.

Description:

Admin address reset is handled by `changeAdmin` function. Multiple issues can occur if the input address is wrong:

- If `admin_` address is set to zero accidentally then it will lock out the contract administrator forever. It is recommended to add a zero address check before setting the address.
- Smart-contract developer can use two-step address verification to avoid setting the wrong address. This means that in the second step, the address that will be set as the administrator must call a function confirming that it was not an incorrect input.

Please note that setting a wrong admin address will result in disabling an important feature of McAfeeDex, which eliminates the taker fees for future whitelisted users.

Code snippet:

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L180#L183~~

2.10. Risk of reentrancy

Severity: low.

Code snippet:

- call

Description:

A `call` method has no gas limit, and it is possible to make reentrancy from another contract. There is no danger in this implementation but this can pose a potential threat.

Other Notes

2.11. Interchain collision

Severity: note.

Description:

This is not a problem with this contract, but it can still lead to financial losses for its customers.

Ethereum addresses are not confirmed on-chain. This means that an abstract address is valid for any Ethereum sisterchain. For the McAfeeDEX contract this means that users can successfully send their funds into the address of the contract at any sisterchain (for example at ETC chain) and the transaction will succeed because there is no contract at ETC chain at the same address and the transaction will not be reverted.

While this is not an issue of this contract, I would recommend to deploy a dummy contract (with ERC20 extraction function) at the most popular Ethereum forks to save users from possible mistakes.

This is not just a theoretical issue! Here is the real example of fund loss.

Recommendation:

I would recommend deploying a dummy contract at the same address at Ethereum fork-chains. The address of the contract is determined by the creator's address and the transaction nonce. So, a developer can use the same address that was used to create the contract at the Ethereum mainnet. It is necessary to increase the nonce at the forkchain to `(deployment nonce - 1)`.

Then you need to deploy the dummy contract. This will cause the dummy contract to be deployed to the same address as the main McAfeeDEX contract.

2.12. Lack of upgrading functions

Severity: note.

Description:
It is advised to design contracts so that it is possible to migrate/upgrade them in the future.

In some specific circumstances, it may be worth to stop users from using the previous version of the contract if some issues were detected. I would recommend implementing a `freeze` function that can stop users from depositing their funds into the contract.

This is possible to implement it so that the function will be disabled after a certain period of time, for example, one year.

2.13. Throw Keyword

Severity: note.

Description:

A `throw` was used multiple times inside the audited contract. Please note that it is deprecated in favor of `revert`, `require`, or `assert` since solidity 0.4.13 version.

Please note that, using `require` in some specific cases will allow returning the users' remaining gas after transaction execution. This will also leave a return message that provides readable reasons about the `throw`.

Refer to Solidity docs for more details.

Recommendation:

Developers can use the latest solidity version for contract development.

Code snippet:

https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/master/contract.sol#L21#L23

https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/master/contract.sol#L119

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/master/contract.sol#L124#L125~~

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/master/contract.sol#L177~~

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/master/contract.sol#L181~~

2.14. Unnecessary Extra Computation

Severity: note.

Description:

`feeRebateXfer` is set to zero and should be removed from `tradeBalances` function since it adds extra complexity and gas usage.

Code snippet:

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L267~~

https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L272

https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L273

2.15. Account Level Contract

Severity: note.

Description:

`AccountLevels` and `AccountLevelsTest` contracts can be removed since only two-level user logic is implemented. It is possible to use `accountLevels` mapping instead.

Code snippet:

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L131~~

~~https://github.com/RideSolo/MCAFEEDEXCONTRACT/blob/5db8389ce23ce11fb273c9573772287aa75ce43b/contract.sol#L139~~

2.16. Anyone can set account level

Severity: note.

Description:

The `AccountLevelsTest` contract member function `setAccountLevel` allow to set any level to any account without owner restriction.

2.17. Unused variable

Severity: note.

Code snippet:

- accountLevelsAddr
- feeRebate

Description:

1. `accountLevelsAddr` is not used anywhere at `SwitchDex` contract. The functionality of `AccountLevels` contract is not used as well.
2. `feeRebate` variable is not used properly.

2.18. Owner Privileges

Severity: owner privileges.

Description:

Contract owner allows himself to:

- mint and burn any amount tokens for any addresshere.
- change Account Levels here.
- change fees here.

3. Conclusion

The audited smart contract must not be deployed. Reported issues must be fixed prior to the usage of this contract.

4. Revealing audit reports

- ~~https://gist.github.com/danbogd/0d69629206d1a4473523edee33fd6cdf~~
- https://gist.github.com/Dexaran/7c170cefa4f06f8dc5cb8c00c2a920ab
- https://gist.github.com/gorbunovperm/d8bae42e359f2de5551f3bfc312c964e
- https://gist.github.com/MrCrambo/f5b5c3f6fb934bd944202e917d7e5561
- https://gist.github.com/RideSolo/4bd626b544ae8f38400046dee67716f6

4.1 Notes about ~~danbogd~~ report.

~~3.8. Bug – rounding error.~~ the `amount` will have a bigger size with decimals as in your example. Therefore no problem in the calculation. Not a security issue.

~~3.9. Truncated division~~ the `amount` is in `amountGet` terms. Therefore it will be truncated the only amount that is less than the lowest decimal. Not a security issue.

4.2 Notes about Dexaran report.

3.1 Lack of ERC20 extraction functions. McAfeeDex has a user interface and does not require direct blockchain transaction with the contract. Also, the McAfeeDex contract address is not published on the `mcafeedex.com` website. If a user intentionally commits an incorrect transaction that is not provided for by the user interface, then this is not a smart contract error. This issue can be Medium severity in maximum. This is a variation of **“Lack of transaction handling mechanism issue.”** that is low severity.

4.3 Notes about MrCrambo report.

3. Anyone can set account level pointed function is a member of `AccountLevelsTest` contract, which is unused. This is Note severity.

Appendix

Smart Contract Audits by Callisto Network.

Miscellaneous

Why Audit Smart Contracts?

Our Most Popular Audit Reports.

Trust the Blockchain, Audit the Smart Contracts.

Follow Callisto's Security Department on Twitter to get our latest news and updates!

Published on **November 10, 2020**

Security Audits

‹ Previous post

Next post ›