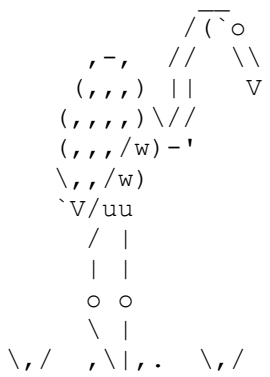
Smart Contract Security Audit

Flamincome

2020-09-16

Introduction

Flamincome is a new DeFi platform to do efficient farmer on *Flamingo*, allowing users to deposit tokens to *flamincome* vaults, where they are allocated to different DeFi protocols/pools with the goal of maximizing their APY.



As requested by **Flamincome** team and as part of the vulnerability review and management process, Red4Sec has been asked to perform a security code audit in order to evaluate the security of the *Normalizer Smart Contract* source code.

Scope

The scope of this evaluation includes:

- NormalizerMethane Smart Contracts.
  - https://github.com/flamincome/contracts/blob/b8aaac5db9653b55fa63428e037e4c7ddf4803ed/implementations/normalizer/NormalizerMethane.sol

Conclusions

To this date, 16th of September 2020, the general conclusion resulting from the conducted audit, is that the *NormalizerMethane Smart Contract* is **not completely secure** and do present some vulnerabilities that could **compromise the security of the users and their information**.

The general conclusions of the performed audit are:

- The small fragment of the audited code represents an extremely limited part of the project; for this reason, it is difficult to contemplate global security, therefore it is impossible to assure the security of the entire project.
- During the security review, the absence of the Unit Test has been detected, which is a highly recommended practice and has become mandatory in projects of this size, destined to manage large amounts of capital.
- One High-risk vulnerability has been detected during the security audit. The security level of the application is improvable and includes hazardous vulnerabilities that should be fixed as soon as possible.
- Some of the detected vulnerabilities do not pose a risk by themselves and have been classified as informative or low risk vulnerabilities. However, it is advisable to apply all the proposed recommendations in order to improve the project source code.

Issues and Recommendations

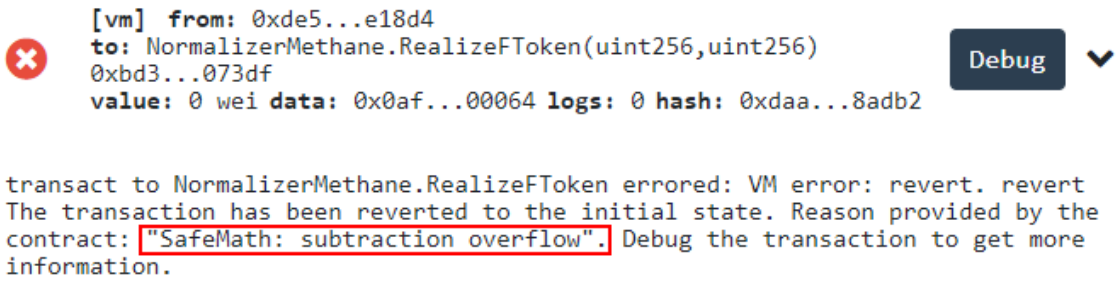
Kill switch (Critical)

It has been verified that the logic related to ***overfillE18*** won't admit values different from zero. This situation causes a denial of service (DoS) controlled by the Owner.

```
function GetMaximumNToken(address _addr) public view returns (uint256) {
    uint256 _val = f[_addr].mul(Vault(address(token)).priceE18());
    uint256 _over = _val.mul(overfillE18);
    return _val.sub(_over).div(1e18);
}
```

Assuming the value 1, the ***_over*** value will contain the exact same value as ***_val*** due to multiplication, this will cause the function to always return 0, avoiding token withdrawal.

In a scenario where the value is greater than 1, such subtraction will trigger an Integer Underflow.



In the following line you can see why the error occurs, *SafeMath* will avoid subtracting from ***_val*** more than the existing amount, producing a *revert* in the transaction which will disable the contract.

```
function GetMaximumNToken(address _addr) public view returns (uint256) {
    uint256 _val = f[_addr].mul(Vault(address(token)).priceE18());
    uint256 _over = _val.mul(overfillE18);
    return _val.sub(_over).div(1e18);
}
```

Source references:

- https://github.com/flamincome/contracts/blob/b8aaac5db9653b55fa63428e037e4c7ddf4803ed/implementations/normalizer/NormalizerMethane.sol#L45

One Person Key (Low)

It is recommended to modify the *NormalizerMethane* governance system to a multi-signature system, decentralized governance or Timelock contract in order to increase the project's trust. Additionally, this will prevent a loss of access in the case of any accident, hack or loss of keys.

It is also recommended to adjust the modification logic of such, to a logic that allows to verify that the new owner is valid and exists.

```
function proposeOwner(address _proposedOwner) public onlyOwner
    {
        require(msg.sender != _proposedOwner, ERROR_CALLER_ALREADY_OWNER);
        proposedOwner = _proposedOwner;
    }

function claimOwnership() public
    {
        require(msg.sender == proposedOwner, ERROR_NOT_PROPOSED_OWNER);
        emit OwnershipTransferred(_owner, proposedOwner);
        _owner = proposedOwner;
        proposedOwner = address(0);
    }
```

Public Variables Naming (Informative)

It is advisable to change the type of the *token* variable from IERC20 to Vault and to rename it *vault*, since conversions (casts) are used throughout the contract, between interfaces that could be avoided. This action will simplify reading and understanding the code.

```
IERC20 public token;
mapping(address => uint256) public f;
mapping(address => uint256) public n;
address public governance;
uint256 public overfillE18;

constructor(address _token)
    public
    ERC20(
        string(
            abi.encodePacked(
                "normalized ",
                ERC20(Vault(_token).token()).name()
            )
        ),
        string(abi.encodePacked("n", ERC20(Vault(_token).token()).symbol()))
    )
{
    _setupDecimals(ERC20(_token).decimals());
    token = IERC20(_token);
    governance = msg.sender;
    overfillE18 = 0;
}
```

Furthermore, it is recommended to rename public variables to more comprehensive names. Since such variables are public, this will make it more user friendly and the contract will be easier for developers to use.

- https://github.com/flamincome/contracts/blob/b8aaac5db9653b55fa63428e037e4c7ddf4803ed/implementations/normalizer/NormalizerMethane.sol#L19-L20
- https://github.com/flamincome/contracts/blob/b8aaac5db9653b55fa63428e037e4c7ddf4803ed/implementations/normalizer/NormalizerMethane.sol#L70
- https://github.com/flamincome/contracts/blob/b8aaac5db9653b55fa63428e037e4c7ddf4803ed/implementations/normalizer/NormalizerMethane.sol#L75

Use of Require statement without reason message (Informative)

Throughout the audit, it has been verified that the reason message is not specified in some of the required methods, in order to give the user more information, which consequently makes it more user friendly.

```
function MintNToken(uint256 _amount) public {
    n[msg.sender] = n[msg.sender].add(_amount);
    require(n[msg.sender] <= GetMaximumNToken(msg.sender));
    _mint(msg.sender, _amount);
}
```

This functionality is compatible since version *0.4.22* and the contract's pragma indicates *0.6.2*, which will result compatible with this feature.