# DefiDollar

## Table of Contents


- Details
- Issues Summary
- Executive summary
  - Documentation
  - Meetings
- Scope
- Issues
  - [IController] - defined as a Contract rather than Interface
  - [Controller] - Emit event when adding a peak / vault
  - [Controller] - Should check if peak is already added before enabling it
  - [Controller, YVaultZap] - Use approve instead of safeApprove
  - [Core] - Optimize mint gas usage
  - [Core] - Apply "Check-Effects-Interactions" at all times
  - [Core] - Remove unused events
  - [YVaultPeak] - Augment get_virtual_price with price feed data from Chainlink
- Artifacts
  - UML Diagram
- License


## Details

- **Client** Yield Studio Pte. Ltd.
- **Date** December 2020
- **Lead reviewer** Daniel Luca (@cleanunicorn)
- **Reviewers** Daniel Luca (@cleanunicorn), Andrei Simion (@andreiashu)
- **Repository**: DefiDollar
- **Commit hash** `8c693e4c9fc0303a1a9dced77412bfa93913c523`
- **Technologies**
  - Solidity
  - Node.JS

## Issues Summary

| SEVERITY | OPEN | CLOSED |
|---|---|---|
| Major | 0 | 0 |
| Medium | 0 | 0 |
| Minor | 0 | 7 |
| Informational | 0 | 1 |

## Executive summary

This report represents the results of the engagement with **Yield Studio Pte. Ltd.** to review **DefiDollar**.

The review was conducted over the course of **~2 weeks** from **November 24 to December 2, 2020**. A total of **7 person-days** were spent reviewing the code.

# Scope

The initial review focused on the DefiDollar identified by the commit hash `8c693e4c9fc0303a1a9dced77412bfa93913c523`.

We were asked to exclude the following folders and contracts:

- contracts/peaks/curve
- contracts/stream/mocks
- contracts/stream/Oracle.sol
- contracts/valley

We focused on manually reviewing the codebase, searching for security issues such as, but not limited to re-entrancy problems, transaction ordering, block timestamp dependency, exception handling, call stack depth limitation, integer overflow/underflow, self-destructible contracts, unsecured balance, use of origin, gas costly patterns, architectural problems, code readability.

**Closer look at**

- Core.sol - has some variables that are redundant now but were a part of the system and can't be deleted without changing the storage layout - so just need to verify that the redundant variables which refer to the contract in the above directories do not have a side affects/exploit scenarios.
- Upgradable Proxy pattern
- Side-effects of deprecated parts for e.g. `peaksAddresses[0]` has now been marked extinct and `peaksAddresses[1]` is the currently used `yVaultPeak`

# Issues

## [IController] - defined as a Contract rather than Interface

Status Fixed Severity Minor

### Description

`IController` is defined as a contract.

code/contracts/interfaces/IController.sol#L5

```
contract IController {
```

Typically, the `ISomething` notation describes an interface. In this case it describes a contract. However it looks like an interface, the functions are not implemented, but merely defined.

code/contracts/interfaces/IController.sol#L6-L9

```
    function earn(address _token) external;
    function vaultWithdraw(IERC20 token, uint _shares) external;
    function withdraw(IERC20 token, uint amount) external;
    function getPricePerFullShare(address token) external view returns(uint);
```

### Recommendation

Use `interface` instead of `contract` when defining `IController`.

## [Controller] - Emit event when adding a peak / vault

Status Fixed Severity Minor

**Description**

Adding a peak is a really important action on the platform.

code/contracts/peaks/Controller.sol#L76-L79

```
    function addPeak(address peak) external onlyOwner {
        require(Address.isContract(peak), "peak is !contract");
        peaks[peak] = true;
    }
```

Similarly, a vault is also important.

code/contracts/peaks/Controller.sol#L81-L85

```
    function addVault(address token, address vault) external onlyOwner {
        require(Address.isContract(token), "token is !contract");
        require(Address.isContract(vault), "vault is !contract");
        vaults[token] = IVault(vault);
    }
```

**Recommendation**

Consider emitting an event when they are added.

**[optional] References**

## [Controller] - Should check if peak is already added before enabling it

Status Fixed Severity Minor

### Description

The method `addPeak` enables a new peak.

code/contracts/peaks/Controller.sol#L76-L79

```
    function addPeak(address peak) external onlyOwner {
        require(Address.isContract(peak), "peak is !contract");
        peaks[peak] = true;
    }
```

The peak is enabled whether the peak was already enabled or not.

### Recommendation

Add a `require` check to make sure the peak is not already enabled.

This can notify the transaction creator before the transaction is executed that it will fail if one peak is already added.

---

## [Controller, YVaultZap] - Use `approve` instead of `safeApprove`

Status Fixed Severity Minor

### Description

The `approve()` method in ERC20's defined tokens suffers from an attack known as a *multiple withdrawal attack*. The method was envisioned as a way for token holders to permit other users and dapps to withdraw a capped number of tokens.

When a token holder wants to adjust the amount of approved tokens from N to M (either increase or decrease), a malicious user or dapp who is approved for N tokens can front-run the adjustment transaction first to withdraw N tokens, then allow the approval to be confirmed, and withdraw additional M tokens.

For this security hole to be exploited, the `approve` and `transfer` methods need to be executed in separate transactions.

Within DefiDollar contracts, the `safeApprove()` method from the `openzeppelin-contracts/SafeERC20` library has been used for allowing different subsystems (both internal, DefiDollar contracts (but also external like Curve LP) to transfer tokens between them.

code/contracts/peaks/Controller.sol#L37-L38

```
            token.safeApprove(address(vault), 0);
            token.safeApprove(address(vault), b);
```

There are 2 issues with the use of this method within DefiDollar:

1. `safeApprove()` has been deprecated in the latest version of openzeppelin-contracts since it's not really solving the *multiple withdrawal attack* correctly:

contracts/token/ERC20/SafeERC20.sol#L30-L37

```
    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
```

2. Since all `approve` and `transferFrom` operations within DefiDollar happen within the same transaction, the risk of a *multiple withdrawal attack* is non-existent.

### Recommendations

Use the plain ERC20 `approve()` method when transferring tokens between contracts within the same transaction.

## [Core] - Optimize mint gas usage

Status Fixed Severity Minor

### Description

The current implementation of `mint` loads the whole `peak` structure in memory before accessing a few properties.

code/contracts/base/Core.sol#L93

```
        Peak memory peak = peaks[msg.sender];
```

The only properties which are accessed are

- `peak.amount`

code/contracts/base/Core.sol#L94

```
        uint tvl = peak.amount.add(dusdAmount);
```

- `peak.state`

code/contracts/base/Core.sol#L97

```
            && peak.state == PeakState.Active
```

- `peak.ceiling`

code/contracts/base/Core.sol#L98

```
            && tvl <= peak.ceiling,
```

However, the whole structure also contains an array, which is not necessary for this method.

code/contracts/base/Core.sol#L37-L42

```
    struct Peak {
        uint[] systemCoinIds; // system indices of the coins accepted by the peak
        uint amount;
        uint ceiling;
        PeakState state;
    }
```

Because the structure is loaded into memory (by specifying `memory` instead of `storage`), the whole structure is loaded from storage and saved into memory. In some cases, this isn't the best option because it can load data that isn't needed, wasting gas.

By comparison, the `storage` keyword will create a reference to the storage data without loading all the data into memory.

The peak amount is also updated in this method.

code/contracts/base/Core.sol#L101

```
        peaks[msg.sender].amount = tvl;
```

### Recommendation

Because it reduces gas usage and because the structure is also modified, the `storage` keyword can be specified when loading the peak.

This can also be applied to `redeem`.

code/contracts/base/Core.sol#L107-L126

```
    /**
    * @notice Redeem DUSD
    * @dev Only whitelisted peaks can call this function
    * @param dusdAmount DUSD amount to redeem.
    * @param account Account to burn DUSD from
    */
    function redeem(uint dusdAmount, address account)
        external
        returns(uint usd)
    {
        Peak memory peak = peaks[msg.sender];
        require(
            dusdAmount > 0 && peak.state != PeakState.Extinct,
            "ERR_REDEEM"
        );
        peaks[msg.sender].amount = peak.amount.sub(peak.amount.min(dusdAmount));
        dusd.burn(account, dusdAmount);
        emit Redeem(account, dusdAmount);
        return dusdAmount;
    }
```

## [Core] - Apply "Check-Effects-Interactions" at all times

Status Fixed Severity Minor

### Description

Not all checks are done before applying the changes. Even if in this case is not a problem, applying the pattern is a good rule of thumb.

code/contracts/base/Core.sol#L67-L77

```
        require(
            address(_dusd) != address(0),
            "0 address during initialization"
        );
        dusd = _dusd;
        stakeLPToken = _stakeLPToken;
        oracle = _oracle;
        require(
            _redeemFactor <= FEE_PRECISION && _colBuffer <= FEE_PRECISION,
            "Incorrect upper bound for fee"
        );
```

### Recommendation

Apply Check-Effects-Interactions pattern.

### References

Solidity docs

## [Core] - Remove unused events

Status Fixed Severity Minor

### Description

There are a few events that don't seem to be used anymore.

code/contracts/base/Core.sol#L50

```
    event FeedUpdated(uint[] feed);
```

code/contracts/base/Core.sol#L53

```
    event UpdateDeficitState(bool inDeficit);
```

### Recommendation

Consider removing these events.

### References

These events seem to be related to the method `whitelistPeak`.

code/contracts/base/Core.sol#L210

```
        bool /* shouldUpdateFeed */
```

# [YVaultPeak] - Augment get_virtual_price with price feed data from Chainlink

Status Acknowledged Severity Informational

**Description**

`yCrvToUsd` function uses Curve's `get_virtual_price` to determine the price of the yCrv token in USD.

code/contracts/peaks/yearn/YVaultPeak.sol#L104-L106

```
    function yCrvToUsd() public view returns (uint) {
        return ySwap.get_virtual_price();
    }
```

This, in turn, is used in most of the user-facing functionality of the YVaultPeak. Therefore any malicious manipulation of price can have wide-reaching consequences in the system.

code/contracts/peaks/yearn/YVaultPeak.sol#L86

```
        _yCrv = dusdAmount.mul(1e18).div(yCrvToUsd()).mul(redeemMultiplier).div(MAX);
```

code/contracts/peaks/yearn/YVaultPeak.sol#L99

```
        _yCrv = dusdAmount.mul(1e18).div(yCrvToUsd()).mul(redeemMultiplier).div(MAX);
```

code/contracts/peaks/yearn/YVaultPeak.sol#L144-L148

```
    function yUSDToUsd() public view returns (uint) {
        return controller.getPricePerFullShare(address(yCrv)) // # yCrv
            .mul(yCrvToUsd()) // USD price
            .div(1e18);
    }
```

code/contracts/peaks/yearn/YVaultPeak.sol#L150-L153

```
    function portfolioValue() external view returns(uint) {
        (,uint total) = yCrvDistribution();
        return total.mul(yCrvToUsd()).div(1e18);
    }
```

Recently, there has been an attack on Compound platform which made use of the fact that their contracts used only one price source feed (Coinbase) that quoted the price of DAI 30% higher vs the USD (Compound DAI Liquidation Event Analysis).

We believe this is the reason why very soon after this attack, Curve published a blog post that recommends using Chainlink price feed data on top of the `get_virtual_price` call: Chainlink Oracles x Curve Pool Tokens

Excerpt from the article:

> Curve pools have so-called virtual price which can be useful for safe pricing of Curve LP tokens. It represents a non-manipulatable USD value of Curve LP tokens. Unfortunately, it doesn't correspond to value in any particular stablecoin though which can be a significant limitation when designing protocols.

**Recommendation**

We believe that an ideal solution for this issue is one that matches DefiDollar as a product and its users. Therefore along with the proposed solution that Curve outlined in their post we want to add our own thoughts for consideration.

Our suggestion trades the availability of some of the functions of the system for a more secure system.

Rather than looping through multiple Chainlink price feeds and picking the lowest price, Defi Dollar could just compare the price returned by `get_price_virtual` with the aggregated price from Chainlink. If this difference exceeds a predetermined, configurable threshold (say 2%) then the system should throw an error, otherwise, the average of these two price feeds will be used.

In case you want more security, you can also use other oracles and aggregate data from all of them, this will however increase gas costs, but will also increase security. In case one of the prices is very different from all the others, your aggregated price oracle will report failure which, in turn, will signal the main system to stop working for a limited time, until the prices fall into place once again.

It depends on how you want to handle this scenario, how much of the cost you want to offset towards the user, if you want to run an oracle of your own, or if you want to ignore this scenario altogether.

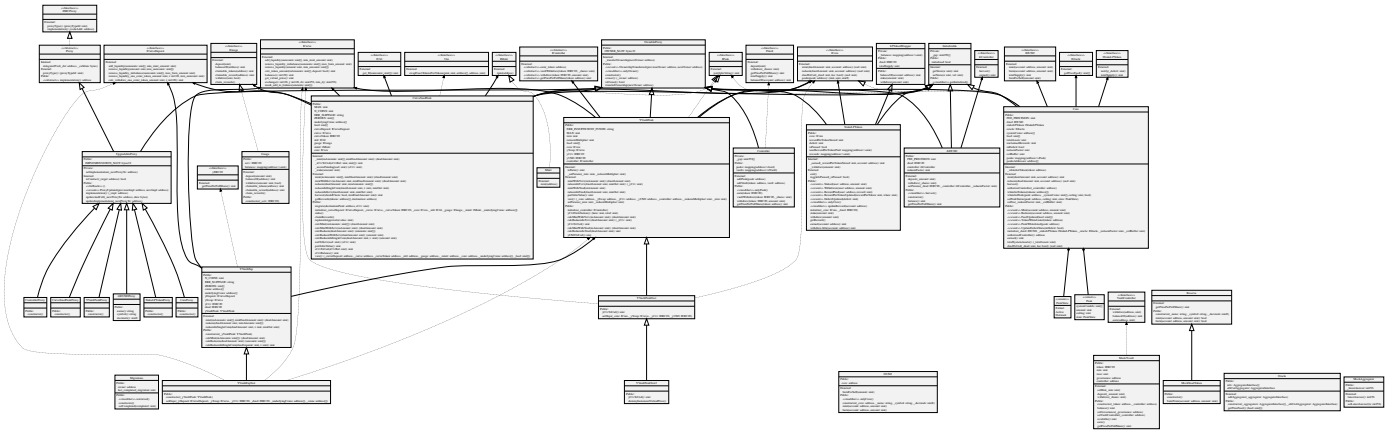
---

# Artifacts

## UML Diagram

Generated with sol2uml:

```
npm link sol2uml --only=production
sol2uml ./code/contracts
```



# License

This report falls under the terms described in the included LICENSE.