MixBytes()

YEARN.FINANCE PROTOCOL V1

SMART CONTRACT

AUDIT REPORT

NOVEMBER 05

2020

TABLE OF CONTENTS

01 INTRODUCTION TO THE AUDIT

GENERAL PROVISIONS

Yearn.Finance is a decentralized investment aggregator that leverages composability and uses automated strategies to earn high yield on crypto assets.

SCOPE OF AUDIT

* https://github.com/iearn-finance/yearn-protocol/tree/9ff0dc0ea73642c529383d0675930a41bf033295/contracts
* https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L517-L1029

02 SECURITY ASSESSMENT PRINCIPLES

CLASSIFICATION OF ISSUES

CRITICAL

Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

MAJOR

Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

WARNINGS

Bugs that can break the intended contract logic or expose it to DoS attacks.

COMMENTS

Other issues and recommendations reported to/acknowledged by the team.

SECURITY ASSESSMENT METHODOLOGY

Two auditors independently verified the code.

Stages of the audit were as follows:

1. “Blind” manual check of the code and its model
2. “Guided” manual code review
3. Checking the code compliance to customer requirements
4. Discussion of independent audit results
5. Report preparation

03 DETECTED ISSUES

CRITICAL

Not found.

MAJOR

Not found.

WARNINGS

1. Unsafe implementation of arithmetic operations

The contractor discovered a code that is potentially susceptible to integer underflow vulnerability. To protect against vulnerabilities of this type, it is recommended to use the SafeMath library which is already contained in the gist submitted for the analysis and is also used in other parts of the code.

https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L787

```
```
function submitClaim(Data storage data, uint coverId) internal returns (uint) {
    Claims claims = Claims(data.nxMaster.getLatestAddress(“CL”));
    claims.submitClaim(coverId);

 ClaimsDataclaimsData=ClaimsData(data.nxMaster.getLatestAddress(“CD”));
    uint claimId = claimsData.actualClaimLength() - 1;
    return claimId;
}
```
```

If the `claimsData.actualClaimLength ()` call returns `0`, then integer underflow will occur and value `uint256 (0-1) = 2 ** 256-1` will be written to the claimId variable. The likelihood of this event to take place directly depends on the implementation of the actualClaimLength method which is not presented in the investigated gist by the Customer.

The contractor understands that the result of calculating actualClaimLength depends on the length of the `allClaims` array (**ClaimsData.sol#L75**), however not using the SafeMath library for safe arithmetic creates additional risks and demonstrates a violation of safe development practices.

Status:

ACKNOWLEDGED

2. Using math operations without the SafeMath safe library

The investigated gist contains a library for the implementation of mathematical operations and protects the contract from integer under/overflow vulnerabilities. However, not all mathematical operations in the contract code are coded using the SafeMath library. It is always recommended to code math operations using SafeMath, unless there is a compelling reason to do otherwise. Moreover, if the logic is implemented without this library, it is important to prevent the occurrence of vulnerabilities such as integer under / overflow. Below several cases are specially given when the Contractor does not see the scenario of vulnerabilities, nevertheless, they are included in the report to emphasize the importance of safe coding of mathematical operations.

https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L912

```
uint expirationTimestamp = block.timestamp + coverAPI.getLockTokenTimeAfterCoverExpiry() + coverPeriod * 1 days;
tokens[nextTokenId] = Token(expirationTimestamp,
    coverCurrency,
    coverDetails[0],
    coverDetails[1],
    coverDetails[2],
    coverDetails[3],
    coverDetails[4],
    coverId, false, 0);
```

Status:

ACKNOWLEDGED

3. Usage of an increment when issuing a token without the Counters safe library

The gist under study already includes the Counters library, which is designed to implement a safe increment and decrement of numerical values. In particular, it is applied in the ERC721 contract (see the lines **https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L156**, **https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L238** and other places), its purpose is to track the number of issued tokens. A similar task of accounting for the number of issued tokens is faced by the Distributor contract, however, the developer preferred not to use the Counters library to mint new tokens in its implementation (see the listing below).

https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L911

```
    // mint token
    uint256 nextTokenId = issuedTokensCount++;
    uint expirationTimestamp = block.timestamp + coverAPI.getLockTokenTimeAfterCoverExpiry() + coverPeriod * 1 days;
    tokens[nextTokenId] = Token(expirationTimestamp,
        coverCurrency,
        coverDetails[0],
        coverDetails[1],
        coverDetails[2],
        coverDetails[3],
        coverDetails[4],
        coverId, false, 0);
    _mint(msg.sender, nextTokenId);
}
```

The Counters library protects integer types from under/overflow vulnerabilities.

Status:

ACKNOWLEDGED

4. Function calculation result is not processed

Below there will be cases when the code does not process the result of calling one or another method. This does not pose a security problem in the given cases but it can be a problem in some circumstances. This is especially true when such methods are called which are defined in extraneous contracts already deployed on the network that are not controlled by the developer.

````
##### Controller.setStrategy(address,address)
```

function setStrategy(address _token, address _strategy) public {
    require(msg.sender == strategist || msg.sender == governance, "!strategist");
    require(approvedStrategies[_token][_strategy] == true, "!approved");

    address _current = strategies[_token];
    if (_current != address(0)) {
      Strategy(_current).withdrawAll();
    }
    strategies[_token] = _strategy;
}
```

contracts/controllers/Controller.sol
````

According to the definition of the Strategy interface:

````
```
    // Controller | Vault role - withdraw should always return to Vault
    function withdrawAll() external returns (uint);
interfaces/yearn/Strategy.sol
```
````

The withdrawAll () method returns the result of its work, which is not processed in the Controller.setStrategy (address, address) method. It depends on the implementation of the withdrawAll () method for a particular strategy.

```
##### Controller.withdrawAll(address)
```

```
function withdrawAll(address _token) public {
    require(msg.sender == strategist || msg.sender == governance, “!strategist”);
    Strategy(strategies[_token]).withdrawAll();
}
```

A similar situation, the result of calling withdrawAl() is not processed in any way.

```
##### Controller.yearn(address,address,uint256)
```

```
    IERC20(_token).safeApprove(onesplit, _amount);
        (_expected, _distribution) = OneSplitAudit(onesplit).getExpectedReturn(_token, _want, _amount, parts, 0);
        OneSplitAudit(onesplit).swap(_token, _want, _amount, _expected, _distribution, 0);
        _after = IERC20(_want).balanceOf(address(this));
contracts/controllers/Controller.sol
```

The swap method returns a value - the result of its work which is not processed in any way:

```
interface OneSplitAudit {
    function swap(
    address fromToken,
    address destToken,
    uint256 amount,
    uint256 minReturn,
    uint256[] calldata distribution,
    uint256 flags
    )
    external
    payable
    returns(uint256 returnAmount);
interfaces/yearn/OneSplitAudit.sol
```

Since the lack of processing of the calculation results is a fairly typical thing for the contracts under study, each case will not be analyzed in detail below.

```
##### yDelegatedVault.repay(address,uint256)
```

The implementation of the yDelegatedVault.repay (address, uint256) method contains lines which lack the processing of the calculation results:

```
function repay(address reserve, uint amount) public  {
    // Required for certain stable coins (USDT for example)
    IERC20(reserve).approve(address(getAaveCore()), 0);
    IERC20(reserve).approve(address(getAaveCore()), amount);
    Aave(getAave()).repay(reserve,                                    amount,
address(uint160(address(this))));
}
```

contracts/vaults/yDelegatedVault.sol

```
##### StrategyMKRVaultDAIDelegate._approveAll()

```
function _approveAll() internal {
    IERC20(token).approve(mcd_join_eth_a, uint(-1));
    IERC20(dai).approve(mcd_join_dai, uint(-1));
    IERC20(dai).approve(yVaultDAI, uint(-1));
    IERC20(dai).approve(unirouter, uint(-1));
}
```
```

The approve method returns true if successful, and may return false otherwise. In fact, it will always return true if the arguments to the function are passed correctly, however, this is yet another case of not processing the calculation results.

Status:

ACKNOWLEDGED

5. No boundary check for `withdrawalFee/performanceFee/strategistReward/keepCRV` values

* **StrategyDForceUSDC.sol#L53-L61**
* **StrategyCreamYFI.sol#L56-L64**
* **StrategyCurveSBTC.sol#L61-L74**
* **StrategyCurveYBUSD.sol#L59-L67**
* **StrategyCurveYCRVVoter.sol#L62-L75**
* **StrategyDForceUSDT.sol#L53-L61**
* **StrategyMKRVaultDAIDelegate.sol#L82-L95**

There are no boundary checks for `withdrawalFee`, `performanceFee`, `strategistReward`, `keepCRV` in setter methods. According to code logic it seems the values of those variables should be less than a particular max constant for each one. We recommend adding a boundary check in the setters to keep the desired invariant.

Status:

ACKNOWLEDGED

6. No boundary check for `min` in yVault

* yVault.sol#L43
* yWETH.sol#L45

There is no boundary check for `min` value. According to code logic it seems the value should be less than defined `max` constant value.

We recommend adding a boundary check in the setters to keep the desired invariant.

Status:

ACKNOWLEDGED

COMMENTS

1. Controller.sol#L98

Check strategy existence for given token.

Status:

ACKNOWLEDGED

2. Controller.sol#L41
StrategyCreamYFI.sol#L77
yDelegatedVault.sol#L96

Use modifiers to check transaction authorization as in the yInsure.sol contract

E.g add modifier `onlyGovernance` and `onlyStrategist`.

Status:

ACKNOWLEDGED

3. Controller.sol#L36
yDelegatedVault.sol#L29-L30

Use constants.

Status:

ACKNOWLEDGED

4. Controller.sol#L67

Use human readable/understandable errors.

Status:

ACKNOWLEDGED

5. yinsure-sol-L517-L1029
yinsure-sol-L561-L595
yinsure-sol-L610-L627

Unused struct ApiId, CurrencyAssets, InvestmentAssets, IARankDetails, McrData.
https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L561-L595

Unused struct Cover, HoldCover.
https://gist.github.com/andrecronje/2b94942709f0594577bec9eca6ed52a0#file-yinsure-sol-L610-L627

Status:

ACKNOWLEDGED

6. yinsure-sol-L768

The line is too long, it’s better to separate to a multiline.

Status:

ACKNOWLEDGED

7. yWETH.sol#L89
   yWETH.sol#L142

We recommend adding reentrancy guards to all public methods which call other contracts just to avoid possible issues.

Status:

ACKNOWLEDGED

8. StrategyCreamYFI.sol#L153

Little bit of wrong line alignment.

Status:

ACKNOWLEDGED

9. StrategyCurveSBTC.sol#L158
   StrategyCurveSBTC.sol#L106
   StrategyCreamYFI.sol#L146
   StrategyCreamYFI.sol#L97
   StrategyCurveYBUSD.sol#L154
   StrategyCurveYBUSD.sol#L99
   StrategyCurveYCRVVoter.sol#L107
   StrategyCurveYCRVVoter.sol#L169
   StrategyCurveYCRVVoter.sol#L140
   StrategyDForceUSDC.sol#L98
   StrategyDForceUSDC.sol#L147

`_fee`/`_keepCRV` potentially can have zero value, so we recommend checking that case to save gas.

Status:

ACKNOWLEDGED

04 | CONCLUSION AND RESULTS

The smart contracts were audited and no critical or major issues were found.

Several suspicious locations have been identified (marked as warning), but it's assumed as not critical. Also we recommend using modifiers and being careful with arithmetical operation(that point about `yinsure.sol`).

05 | YEARN.FINANCE PROTOCOL DEV TEAM COMMENTS

* Yearn Protocol next version will address the return values processing since it changes design on how vault + strategies interact.
* Next version of YInsure will take into account the comments on using SafeMath for the calculations suggested.

APPENDIX TO REPORT

1. External openzeppelin library check for yinsure-sol-L517-L1029

The gist provided by the Customer contains many contracts and libraries imported from the well-known and trusted due to the attention of the community library OpenZeppelin (**https://github.com/OpenZeppelin/openzeppelin-contracts.git**). The code related to the library version v2.5.1 and to commit 837828967a9831e4337deb24fb2009 is used.

If we discard the interfaces and consider directly the code that can be published on the Ethereum network, then the Distributor contract stands out which is the implementation of the ERC721 token and supplemented with its own functionality. One can highlight the NexusMutualCover library, which is used by the Distributor contract. The Distributor contract is inherited from the ERC721Full, Ownable and ReentrancyGuard contracts, used in its code other contracts, interfaces and libraries published in the gist.

The table below shows the contracts, interfaces and libraries borrowed from OpenZeppelin:

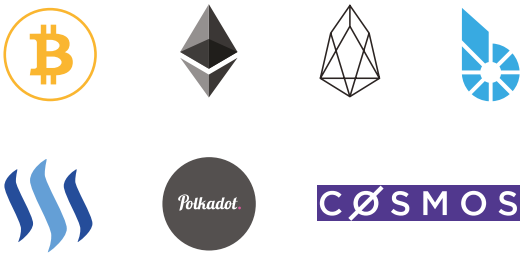
Context	openzeppelin-contracts/contracts/GSN/Context.sol
IERC165	openzeppelin-contracts/contracts/introspection/IERC165.sol
IERC721	openzeppelin-contracts/contracts/token/ERC721/IERC721.sol
IERC721Receiver	openzeppelin-contracts/contracts/token/ERC721/IERC721Receiver.sol
SafeMath	openzeppelin-contracts/contracts/math/SafeMath.sol
Address	openzeppelin-contracts/contracts/utils/Address.sol
Counters	openzeppelin-contracts/contracts/drafts/Counters.sol
ERC165	openzeppelin-contracts/contracts/introspection/ERC165.sol
ERC721	openzeppelin-contracts/contracts/token/ERC721/ERC721.sol
IERC721Enumerable	openzeppelin-contracts/contracts/token/ERC721/IERC721Enumerable.sol
ERC721Enumerable	openzeppelin-contracts/contracts/token/ERC721/ERC721Enumerable.sol
IERC721Metadata	openzeppelin-contracts/contracts/token/ERC721/IERC721Metadata.sol
ERC721Full	openzeppelin-contracts/contracts/token/ERC721/ERC721Full.sol
IERC20	openzeppelin-contracts/contracts/token/ERC20/IERC20.sol
Ownable	openzeppelin-contracts/contracts/ownership/Ownable.sol
ReentrancyGuard	openzeppelin-contracts/contracts/utils/ReentrancyGuard.sol

ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, consult universities and enterprises, do research, publish articles and documentation.

Stack

Blockchains



JOIN US