SFXDX

XBE.finance: yield farming smart-contract audit report

By: SFXDX Team
For: XBE Finance

www.sfxdx.com

XBE: yield farming
smart-contract audit report

The following Audit was prepared by the Sfxdx team and the purpose of this audit is to analyze the codebase of ten smart contracts for quality of the code, security threats and compliance with the business requirements.

Authors:
Maxim Prishchepo,
Stanislav Golovin.

This document describes the compliance of XBE.finance platform`s Smart Contracts with the logical, architectural and security requirements. The following Audit was prepared by the Sfxdx team and the purpose of this audit is to analyze the codebase of ten smart contracts for quality of the code, security threats and compliance with the business requirements.

During the audit, we classified the shortcomings into four categories of severity:

- Informational - shortcomings that are described in the document for informational purposes only. They are not required for correction, do not violate the intended architecture and business logic.

- Low - shortcomings that do not violate the requirements, but can be eliminated to improve the quality of the code. Their elimination can be considered as a recommendation.

- Medium - shortcomings that are not serious, but should be better eliminated. Their elimination can be considered as a recommendation.

- High - shortcomings that lead to the fact that the smart contract does not meet the requirements. For example, business logic violation, inconsistency with the planned architecture etc. This type of shortcomings must be eliminated from the smart contract code.

Based on the results of the audit, the following conclusion can be made:

Generally the implemented business logic and architecture of smart contracts meets initial requirements. Mostly informational shortcomings were found. For example, lack or extra checks.
Few medium or high - severity issues were found.
Also few low-severity shortcomings were detected and eliminating them can improve the code's quality.

A more detailed description of the found issues is provided in the document.
The document is attached.

The xbe.finance yield farming smart-contract audit report

Introduction

Git repository with contracts:

https://github.com/EURxbfinance/vaults

Audited contracts:

EURxbVault.sol
Controller.sol
Registry.sol
Treasury.sol
TokenWrapper.sol
UnwrappedToWrappedTokenConverter.sol
WrappedToUnwrappedTokenConverter.sol

Below are summary conclusions regarding the contract codebase considering the business logic analysis, correctness of logic implementation, security analysis, gas efficiency analysis and style guide principles.

As an appendix to this report the “Test Coverage” report and “Static Analysis” report are also attached.

EURxbVault

09.06.2021

https://github.com/EURxbfinance/vaults/commit/684d8c6d61db3ff2543e19137d14e9eb49f6eb52

Based on results of the audit smart contract “EURxbVault” the following categories of issues were found:

Severity		Count
Informational	-	1
Low	-	5
Medium	-	0
High	-	0

Smart contract EURxbVault can be used in case the award will be given in the same token which staked in Vault

Low

When the user makes the liquidity deposit to the smart contract (a call the function deposit or depositAll), the contract keeps the proportion of the user’s deposit in relation to the deposit amount of all users and general award which was earned by the strategy until the moment of making deposit:
shares = (_amount.mul(totalSupply())).div(_pool);

where *_amount* - amount of tokens deposited by user, *totalSupply()* - total amount of Vault's LP tokens issued to all users BEFORE the current deposit, a *_pool* - the total amount sum of previously contributed tokens by all users and total award which was earned by the strategy until the current moment of making deposit. To sum up: if the amount of awards tokens and tokens which are staked in Vault are not equal, the calculation logic will be subverted.

Inheritance from Initializable contract without using the upgradeable contracts or proxy contracts logic

Low

The "EURxbVault" contract is inherited from the Initializable contract of OpenZeppelin library https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol, which is intended to move the constructor logic into an initialization function. In the OpenZeppelin library this contract is the component of proxy and upgradable contracts concept. However in this implementation the contract is intended for lazy initialization which can be misleading. The lazy initialization creates the flexibility in use by allowing the contract initialization not at the deploy time but later. In this case there is the danger that not initialized contract will be used inappropriately.

Uninformative events: Deposit, Withdraw

Low

The events (Deposit, Withdraw), that did not in the contract before, were added to the EURxbVault contract:
event Deposit(uint256 indexed _shares);
event Withdraw(uint256 indexed _amount);

These events were written for the contract debugging and have to be deleted in the release version or changed. In these events the indexing is used incorrectly because there is no point to index by the number of tokens. Also in both events there is no information about the user who called the corresponding functions.

Not used logic of available function

Low

The yearn repository yVault contract

(https://github.com/yearn/yearn-protocol/blob/develop/contracts/vaults/yVault.sol), which was taken as the basis for the EURxbVault contract, implements the logic of funds borrowing and allows borrowing part of the funds (by default 95% of the contract balance):

```
function available() public view returns(uint256) {
    return _token.balanceOf(address(this)).mul(min).div(max);
}
```

But this logic was not reflected in the other contracts, so the available function received redundant multiplication and division logic. This implementation with unused logic is non-optimal by gas cost.

Redundant variables in _withdraw and _deposit functions

Low

In the current implementation variable _to is redundant because the _withdraw and _deposit functions are internal. Also both functions have _to parameter which take the value _msgSender(). This is another implementation of functions with redundant logic that is non-optimal by gas cost.

The harvest function, which implements logic of the loan repayment by third-party funds, has been removed

Informational

The EURxbVault contract is based on yVault contract of yearn repository (https://github.com/yearn/yearn-protocol/blob/develop/contracts/vaults/yVault.sol). This contract (EURxbVault) implements the logic of the loan repayment possibility by sending third-party funds to the controller address. This functionality is not needed in the current implementation of Vault and it has been removed.

Controller

10.06.2021

https://github.com/EURxbfinance/vaults/commit/e9c8f33be021874e488d7bbd5a85ea3964b8ae28

Based on results of the audit smart contract “Controller” the following categories of issues were found:

Severity		Count
Informational	-	5
Low	-	3
Medium	-	2
High	-	0

The accrual percentages logic of variable "split" has been changed

Medium

The yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract, implements the logic of sending a reward part received from strategies to the Treasury contract, in order to further distribute these funds to the stakers of the Governance contract as a reward for voting. By default the reward part was 5% and this parameter could be changed by voting of the Governance contract. However the function of percentage changing has been removed from the code.

That is why in the current implementation of the Controller contract, it is impossible to change the percentage of income.

There is no condition in the withdraw function that only Vault contract can call it

Medium

There was the using restriction of the withdrawal function of the yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract. The function could be called only by Vault contract. This logic has been removed and now the function can be called by anyone. This is an obvious security violation: an intruder can call the withdraw function at any time and for any amount in order to manipulate users' deposits.

The safeTransfer support has been removed

Low

The contract code includes, but does not use the SafeERC20 library code. Thus Controller will not support the forwarding of ERC 20 tokens of older versions (USMT is the most famous such token).

A possible solution to this problem is to replace all constructions of the form: *IERC20(_token).transfer* to the use constructions of the form: *IERC20(_token).safeTransfer* of SafeERC20 library.

withdrawAll function have been removed

Low

The withdrawAll function in the yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract, has been removed.

The withdrawal function implemented the withdrawal of all funds of all users from the strategy. The governance or strategist address can call this function. The withdrawal function has an important role in the emergency prevention of financial leaks in case of an exploit being detected in contracts. Removing this function complicates the withdrawal, and as a result, the preservation of user funds.

The part parameter of the harvest function has been transferred to the contract variable

Low

There was the yearn function (which has now been renamed harvest) in the yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract. In this function there was the parts parameter that was responsible for the size of the reward part that would be converted from the strategy token to the strategy reward token *IStrategy(_strategy).want()*. Thus, the yearn (harvest) function had a flexible tool of partial conversion of reward tokens for different purposes. Now this logic has been transferred to the contract code and to change this parameter the decision must be made through Governance, which is a much less flexible approach than it was originally.

Inheritance from Initializable contract without using the upgradeable contracts or proxy contracts logic

Informational

The “Controller” contract is inherited from the Initializable contract of OpenZeppelin library https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol, which is intended to move the constructor logic into an initialization function. In the OpenZeppelin library this contract is the component of proxy and upgradable contracts

concept. However in this implementation the contract is intended for lazy initialization which can be misleading.

The lazy initialization creates the flexibility in use by allowing the contract initialization not at the deploy time but later. In this case there is the danger that not initialized contract will be used inappropriately.

Added WithdrawToVaultAll, Earn, Harvest events

Informational

The events, that did not in the contract before, were added to the Controller contract
event WithdrawToVaultAll(address _token);
event Earn(address _token, uint256 _amount);
event Harvest(address _strategy, address _token);

Events are called in functions of the same name and created for debugging, but in general they can be useful, since they are called in key functions of the contract responsible for funds transferring.

The contract initialization logic has been changed (now it is necessary to set the onisplit address after the contract deploy)

Informational

The yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract, had the hardcoded parameter
onesplit = address(0x50FDA034C0Ce7a8f7EFDAebDA7Aa7cA21CC1267e);
that was set in the constructor. This guaranteed the ability to use the Controller contract without having to call an additional method which sets the address of IOneSplitAudit (1Inch protocol interface). The parameter was removed from the constructor so that the Controller contract could work correctly. Now it is necessary to call the setOneSplit function, in which this parameter is set. Only the governance address can call this function.

Removed the functions balanceOf, getExpectedReturn

Informational

The yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract, had functions (balanceOf, getExpectedReturn) which have been removed.

The balanceOf function showed the tokens amount on the selected token's strategy contract. This function was informational and its removal does not affect the functioning of the contract.

The getExpectedReturn function showed the expected number of tokens that will be received as a result of the exchange on the 1Inch protocol. This function was informational and its removal does not affect the functioning of the contract. However, this function made it easier to use this contract showing information that is difficult to calculate for the user.

The approveStrategy/revokeStrategy functions have been replaced by one setApprovedStrategy function

Informational

The yearn repository Controller contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/controllers/Controller.sol), which was taken as the basis for the Controller contract, had two separate functions for setting and canceling the new strategy for the Vault. Now these two functions have been replaced by one. This does not affect the security or operability of the code.

Registry

10.06.2021

https://github.com/EURxbfinance/vaults/commit/e9c8f33be021874e488d7bbd5a85ea3964b8ae28

Based on results of the audit smart contract “Registry” the following categories of issues were found:

Severity		Count
Informational	-	3
Low	-	1
Medium	-	0
High	-	0

Not optimal refactoring addVault, addWrappedVault, add Delegate d Vault, setVault (renamed to _addVault)

Low

The addWrappedVault and addDelegatedVault functions call the public addVault function which calls _addVault. Instead of that _addVault could be called immediately, as was implemented in the ”YRegistry" contract of the yearn repository, which was taken as the basis of this contract: (https://github.com/yearn/yearn-protocol/blob/develop/contracts/registries/YRegistry.sol). As a result of this refactoring, the addWrappedVault and addDelegatedVault functions' gas cost became higher, while the logic of their work remained the same.

Inheritance from Initializable contract without using the upgradeable contracts or proxy contracts logic

Informational

The “Registry” contract is inherited from the Initializable contract of OpenZeppelin library https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol, which is intended to move the constructor logic into an initialization function. In the OpenZeppelin library this contract is the component of proxy and upgradable contracts concept. However in this implementation the contract is intended for lazy initialization which can be misleading.

The lazy initialization creates the flexibility in use by allowing the contract initialization not at the deploy time but later. In this case there is the danger that not initialized contract will be used inappropriately.

Removed the logic of the pending Governance

Informational

The yearn repository “YRegistry” contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/registries/YRegistry.sol), which was taken as the basis for the Registry contract had the logic of changing the governance contract in several stages:

- calling setPendingGovernance function with setting the new governance address
- calling acceptGovernance function on behalf of the new governance address, confirming the change of governance.

This mechanism allowed to change governance safely without fear of specifying an incorrect or erroneous address. This logic was removed and replaced with the usual OnlyOwner functions (only renamed to onlyGovernance).

Removed the ability to change the type of the vault (setWrappedVault, setDelegatedVault functions)

Informational

The yearn repository “YRegistry” contract (https://github.com/yearn/yearn-protocol/blob/develop/contracts/registries/YRegistry.sol), which was taken as the basis for the Registry contract had the logic of changing the vault type in case vault was added with incorrect type. This was done to reduce the influence of the human factor when setting new vaults. However, the functions responsible for the task of the types of vaults: setWrappedVault, setDelegatedVault were removed from the code.

Treasury

07.06.2021

https://github.com/EURxbfinance/vaults/commit/69fae7f56580d7bef5cca1e9fd64011b2e9d17fb

Based on results of the audit smart contract “Treasury” the following categories of issues were found:

Severity		Count
Informational	-	1
Low	-	0
Medium	-	0
High	-	0

Inheritance from Initializable contract without using the upgradeable contracts or proxy contracts logic

Informational

The “Registry” contract is inherited from the Initializable contract of OpenZeppelin library https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/utils/Initializable.sol, which is intended to move the constructor logic into an initialization function. In the OpenZeppelin library this contract is the component of proxy and upgradable contracts concept. However in this implementation the contract is intended for lazy initialization which can be misleading.

The lazy initialization creates the flexibility in use by allowing the contract initialization not at the deploy time but later. In this case there is the danger that not initialized contract will be used inappropriately.

TokenWrapper

01.04.2021

https://github.com/EURxbfinance/vaults/commit/5ba58d317cd4302ea136b0d698dedc3bf0df18f2

Based on results of the audit smart contract “TokenWrapper” the following categories of issues were found:

Severity		Count
Informational	-	0
Low	-	1
Medium	-	0
High	-	1

The TokenWrapper deployer has unlimited rights to mint tokens

High

TokenWrapper contract is inherited from the ERC20PresetMinterPauser, which is inherited from AccessControl and sets up DEFAULT_ADMIN_ROLE in the constructor: _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());

Thus, whoever deployed the contract has unlimited possibilities to create the TokenWrapper to any address. If the account’s private key is compromised, then the intruder may be able to gain access to all the invested funds of the institutional vault (by creating an infinite TokenWrapper balance for his own).

A possible solution to this problem is to abandon the ERC20PresetMinterPauser as a base contract.

The inflationary and deflationary tokens support lack

Low

The TokenWrapper contract InstitutionalEURxbVault is used in InstitutionalEURxbVault for possible creation of two vaults (InstitutionalEURxbVault и ConsumerEURxbVault) for the EURxb token. However, the TokenWrapper logic mints and burns the same token number regardless of time. Thus the permanent inflation of the EURxb token will have a negative impact on the contract. If non-inflation ERC20 regular token will be used as EURxb, there will not be a security issue.

UnwrappedToWrappedTokenConverter

01.04.2021

https://github.com/EURxbfinance/vaults/commit/b3d47dd0799d0d727b441f1bb9ed91581e1a1625

Based on results of the audit smart contract “UnwrappedToWrappedTokenConverter” the following categories of issues were found:

Severity		Count
Informational	-	0
Low	-	2
Medium	-	0
High	-	0

The strategy token must be inheritance from TokenWrapper

Low

Bringing *TokenWrapper(IStrategy(_strategy).want())* type restricts the token choice since the mint logic of TokenWrapper is specific and the conversion function *convert* will work correctly only with this function implementation.

Not optimal convert implementation

Low

There is no parameter which indicates the address to the new tokens will be minted in the TokenWrapper mint function. These tokens are always minted to the address who calls the

function. Thus, inside the convert function of UnwrappedToWrappedTokenConverter contract it is necessary to make a double transfer of tokens, instead of one:

```
wrapper.mint(tokenBalance);
wrapper.safeTransfer(msg.sender, tokenBalance);
```

Firstly the tokens are minted to the UnwrappedToWrappedTokenConverter contract address and then transferred to the calling address.

A possible solution to this problem: to add to the mint function a parameter which indicates to whose address the new tokens will be minted.

WrappedToUnwrappedTokenConverter

01.04.2021

https://github.com/EURxbfinance/vaults/commit/b3d47dd0799d0d727b441f1bb9ed91581e1a1625

Based on results of the audit smart contract “WrappedToUnwrappedTokenConverter” the following categories of issues were found:

Severity		Count
Informational	-	0
Low	-	1
Medium	-	0
High	-	1

Double waste in the convert function

High

The convert function implements the burning of TokenWrapper tokens and transferring Unwrapped tokens to who calls the function:

wrapper.burn(wrappedtokenBalance);
IERC20(token).safeTransfer(msg.sender, wrappedtokenBalance);
However the transfer of Unwrapped tokes to the caller implements inside burn function of TokenWrapper:

```
function burn(uint256 _amount) public override onlyMinter {
        _burn(msg.sender, _amount);
        IERC20(wrappedToken).safeTransfer(msg.sender, _amount);
 }
```

Thus, tokens are transferred twice. A possible solution to this problem: to remove one of the tokens’ transfers.

The strategy token must be inheritance from TokenWrapper

Low

Bringing *TokenWrapper(IStrategy(_strategy).want())* type restricts the token choice since the mint logic of TokenWrapper is specific and the conversion function *convert* will work correctly only with this function implementation.