CERTIK

Xend.Finance: Yearn–Dai

Smart Contracts

Security Assessment

January 27th, 2021

By:
Sheraz Arshad @ CertiK
sheraz.arshad@certik.org

Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Xend.Finance: Yearn-Dai
Description	The repository allows investing of Dai in yearn finance and allows withdrawal. It provides interface through an adapter-service pattern where the service contract serves as main interacting contract for the functionality.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. bd8d35fb8d5919cb97e288dc7887aa7c6b8d29f5 2. e8bcbb45b88fda634c54b801375ff019aaa0fab0

Audit Summary

Delivery Date	January 27th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	November 17th, 2020 - January 27th, 2021

Vulnerability Summary

Total Issues	24
🔴 **Total Critical**	2
🟠 **Total Major**	0
🟡 **Total Medium**	1
🔵 **Total Minor**	11
🟢 **Total Informational**	10

Executive Summary

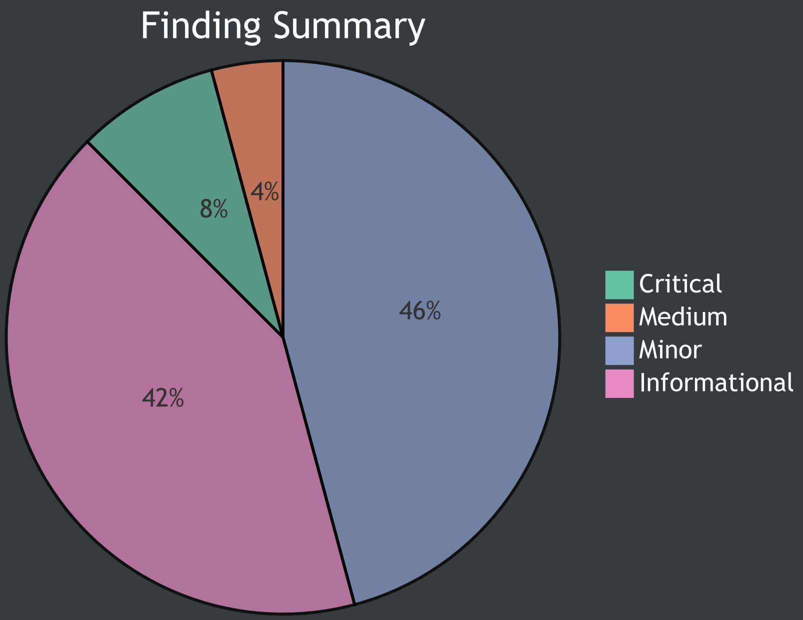
This report represents the results of CertiK's engagement with Xend on their implementation of the Yearn-dai contracts.

Our findings mainly refer to optimizations and a couple of major issues. All of the findings except a few informational were remediated. The overall security of the contracts can be deemed as high after the remediations were applied.

Files In Scope

ID	Contract	Location
DLA	DaiLendingAdapter.sol	DaiLendingAdapter.sol
DLS	DaiLendingService.sol	DaiLendingService.sol
IDT	IDaiToken.sol	IDaiToken.sol
IYD	IYDaiToken.sol	IYDaiToken.sol
IDL	IDaiLendingService.sol	IDaiLendingService.sol
OWN	Ownable.sol	Ownable.sol

Findings



ID	Title	Type	Severity	Resolved
OWN-01	`if` block should be substitued with `require` call	Coding Style	● Minor	✓
OWN-02	`if` block can be substituted with a `require` call	Coding Style	● Minor	ⓘ
OWN-03	Unlocked Compiler Version	Language Specific	● Informational	✓
DLS-01	Unlocked Compiler Version	Language Specific	● Informational	✓
DLS-02	`if` block can be substituted with a `require` statement	Coding Style	● Minor	ⓘ

DLA-01	Unlocked Compiler Version	Language Specific	🟢 Informational	✓
DLA-02	Mutability Specifiers Missing	Gas Optimization	🟢 Informational	✓
DLA-03	Inefficient code	Coding Style	🟢 Informational	✓
DLA-04	`storage` is updated after external interaction	Coding Style	🟡 Medium	✓
DLA-05	Function visibility can be changed to `external`	Gas Optimization	🟢 Informational	✓
DLA-06	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-07	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-08	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-09	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-10	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-11	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-12	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-13	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
DLA-14	Inefficient code	Gas Optimization	🟢 Informational	✓
IDT-01	Incorrect code	Logical Issue	🔴 Critical	✓
IDT-02	Unlocked Compiler Version	Language Specific	🟢 Informational	✓
IYD-01	Unlocked Compiler Version	Language	🟢	✓

		Specific	Informational	
IDL-01	Unlocked Compiler Version	Language Specific	🟢 Informational	✓
IDL-02	Function signature in `interface` not declared external	Compiler Error	🔴 Critical	✓

OWN-01: `if` block should be substitued with `require` call

Type	Severity	Location
Coding Style	● Minor	Ownable.sol L32-L34

Description:

The `if` block on the aforementioned line evaluates to `false` when an zero address is provided yet the transaction executes successfully with setting a new contract owner.

Recommendation:

We recommend to use a `require` function such that the transaction reverts when an zero address is provided to increase the legibility of the code.

```
    require(newOwner != address(0), "address cannot be zero");
```

Alleviation:

Alleviations were applied as advised.

OWN-02: `if` block can be substituted with a `require` call

Type	Severity	Location
Coding Style	● Minor	Ownable.sol L41-L43

Description:

The `if` block on the aforementioned line evaluates to `false` when an zero address is provided yet the transaction executes successfully with setting a new contract owner.

Recommendation:

We recommend to use a `require` function such that the transaction reverts when an zero address is provided to increase the legibility of the code.

```
require(newServiceContract != address(0), "address cannot be zero");
```

Alleviation:

No alleviations.

OWN-03: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	Ownable.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised.

DLS-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	DaiLendingService.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised.

DLS-02: `if` block can be substituted with a `require` statement

Type	Severity	Location
Coding Style	● Minor	DaiLendingService.sol L14

Description:

The `if` block on the aforementioned line evaluates to `false` when an zero address is provided yet the transaction executes successfully with setting a new contract owner.

Recommendation:

We recommend to use a `require` function such that the transaction reverts when an zero address is provided to increase the legibility of the code.

```
require(_owner != address(0), "address cannot be zero");
```

Alleviation:

No alleviations.

DLA-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	DaiLendingAdapter.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised.

DLA-02: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	● Informational	DaiLendingAdapter.sol L74, L76

Description:

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor` 's execution.

Recommendation:

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation:

Alleviations were applied as advised.

DLA-03: Inefficient code

Type	Severity	Location
Coding Style	● Informational	DaiLendingAdapter.sol L135-L140

Description:

The code on the aforementioned lines in `GetNetRevenue` is redundant as the same calculation is performed by the function `GetGrossRevenue` .

Recommendation:

We recommend to utilize the call to function `GetGrossRevenue` in place of the aforementioned lines in `GetNetRevenue` to reduce bytecode footprint of the contract which will result in reduced deployment gas cost and it also increases the legibility of the codebase.

Alleviation:

Alleviations were applied as advised.

DLA-04: `storage` is updated after external interaction

Type	Severity	Location
Coding Style	● Medium	DaiLendingAdapter.sol L166-L170, L200-L204, L236-L242

Description:

The aforementioned lines contain code blocks which update `storage` after external interactions are performed which can open doors to re-entrancy attacks. Additionally, the `dai` and `yDai` tokens do not have constant addresses within the current codebase, re-entrancy is still a possibility in the case that the token implementations change to a malicious implementation.

Recommendation:

We recommend to update `storage` before the external interactions or `ReentrancyGuard` contract from OpenZeppelin can be used and `nonReentrant` modifier can be added to the signatures of vulnerable functions.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol

Alleviation:

Alleviations were applied as advised.

DLA-05: Function visibility can be changed to `external`

Type	Severity	Location
Gas Optimization	● Informational	DaiLendingAdapter.sol L85, L89, L93, L100, L133, L147, L181, L218

Description:

The functions on the aforementioned lines are never called from within the contract and hence their visibilities can be changed to `external`.

Recommendation:

We recommend to change the visibility of the functions on the aforementioned lines from `public` to `external` as they are never called from within the contract.

Alleviation:

Alleviations were applied as advised.

DLA-06: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	DaiLendingAdapter.sol L114

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-07: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	DaiLendingAdapter.sol L157

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-08: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	DaiLendingAdapter.sol L163

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom() / safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-09: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	DaiLendingAdapter.sol L191

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-10: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	🔵 Minor	DaiLendingAdapter.sol L197

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-11: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	🔵 Minor	DaiLendingAdapter.sol L225

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-12: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	DaiLendingAdapter.sol L233

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-13: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	DaiLendingAdapter.sol L269

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances through the use of `safeTransferFrom()` / `safeTransfer()` functions of `SafeERC20` library.

Alleviation:

Alleviations were applied as advised.

DLA-14: Inefficient code

Type	Severity	Location
Gas Optimization	● Informational	DaiLendingAdapter.sol L147, L181, L218

Description:

The `Withdraw` , `WithdrawByShares` and `WithdrawBySharesOnly` functions contain code duplication and inefficient `userDaiDeposit` mapping lookups.

Recommendation:

It would be advisable to make an internal `_withdraw` function, which has parameters and the capability to support the functionality for all three public-facing withdraw functions, as well as looking up `userDaiDeposits[owner]` only once, storing it in a local variable, and referencing that local variable instead of subsequent lookups of `userDaiDeposits[owner]` .

Alleviation:

Alleviations were partly applied as advised.

IDT-01: Incorrect code

Type	Severity	Location
Logical Issue	● Critical	IDaiToken.sol L1

Description:

The file expects to contain interface for `Dai Token` yet it contains interface for `IYDaiToken` .

Recommendation:

We recommend to replace the `IYDaiToken` with `IDaiToken` in the file.

Alleviation:

Alleviations were applied as advised.

IDT-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	IDaiToken.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised.

IYD-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	IYDaiToken.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised.

IDL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	IDaiLendingService.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Alleviations were applied as advised.

IDL-02: Function signature in `interface` not declared external

Type	Severity	Location
Compiler Error	● Critical	IDaiLendingService.sol L20

Description:

An interface can only have function signatures with visibilities specified as `external` yet the function signature on the aforementioned line does not have its visibility specified.

Recommendation:

We advise to add the visibility of `external` to function signature on the aforementioned line.

Alleviation:

Alleviations were applied as advised.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.