XDeFi II

Security Assessment

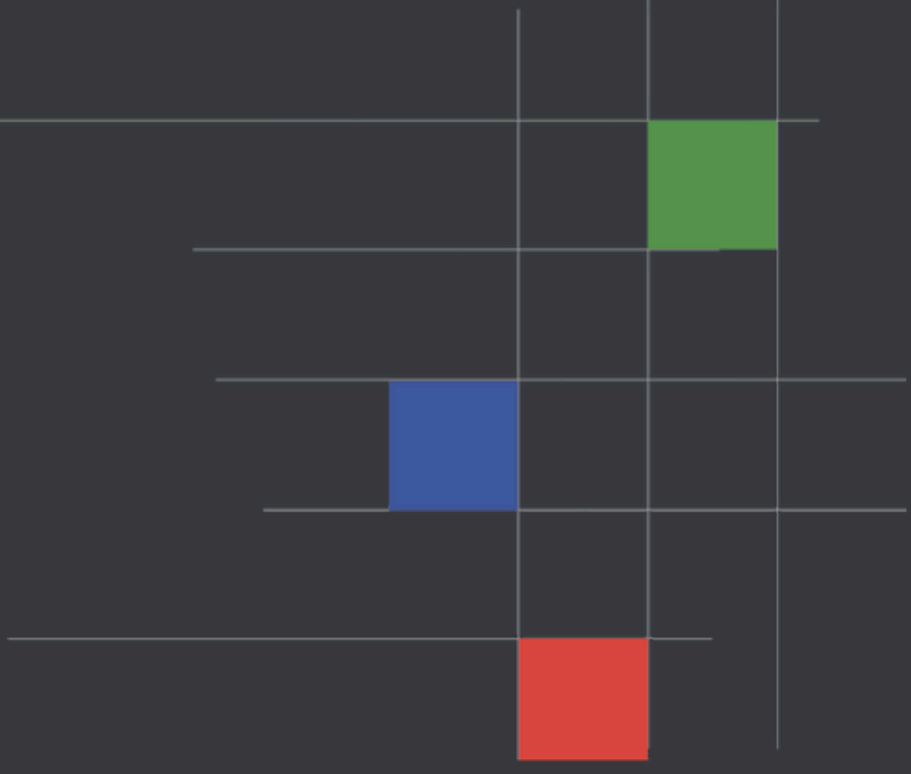
February 26th, 2021

For :
XDeFi

By :

Zach Zhou @ CertiK
jun.zhou@certik.org

Rudolph Wang @ CertiK
shaozhong.wang@certik.org



Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

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	XDeFi II
Description	XDeFi Yield Farming & XDEX on Ethereum
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	58ebea6c64bd5c6aff2a434cd3cf63dfcb13ee27

Audit Summary

Delivery Date	Feb. 13th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 8th, 2021 - Feb. 13th, 2021

Vulnerability Summary

Total Issues	12
Total Critical	0
Total Major	1
Total Minor	2
Total Informational	9

Executive Summary

This report has been prepared for **XDeFi** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

External files `AddressHelper.sol` and `XNum.sol` are imported into some scope files. This audit is based on their safe implementation.

File in Scope

ID	Contract	SHA-256 Checksum
FM	FarmMaster.sol	35964317d9c6c7ca6f7914d20874a5c757f6e418b5ca30e077067838d1c6d873
MG	Migrations.sol	8a6b38936c738a0e612391ee231f39352cc8878f4a5b41c05f0895fb662b3fd6
TL	Timelock.sole	9edaa4bd3de00e5a4e733949577f097b1418f2d12309b9f7810e94a659b2c36b
XD	XDEX.sol	e5c4d2de1b0ca5cf7c8017e627f3dcaf30e8b678957b19d67278a8d0bccd8830
XS	XdexStream.sol	5e8ed0926a2f9fbee8026857fa7646f742c93523877ac3cc8ede77ba9a4e85cd
XH	XHalfLife.sol	83f4b245054d55dffe82d43e9502adbe3711e04ebbb0edf9619c1d5e3cf1a36d
IER	IERC20.sol	20f07b47212ef769e60247eb2118cd4eba7190d8445f735586f88ee9e896bed6
IXH	IXHalfLife.sol	d156b4a460118a038330d30215d0daf5676b27e4e797f5c8eb66b4d9b922bdcf

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **XDeFi** team or reported an issue.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 1 Major vulnerability and 2 Minor vulnerabilities were identified during our audit that solely concerns the specification.

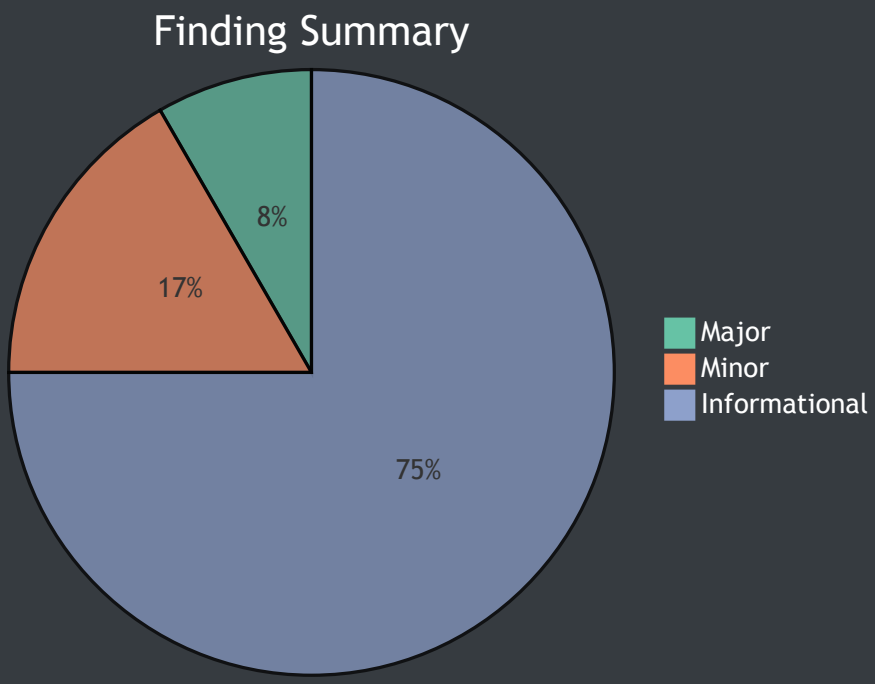
Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, and the code comment coverage is good.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings



ID	Title	Type	Severity	Resolved
FM-01	Missing Emit Events	Optimization	Informational	✓
FM-02	Missing Some Important Checks	Logical Issue	Informational	✓
FM-03	Proper Usage of "public" and "external" Type	Gas Optimization	Informational	✓
FM-04	State Variables that could be Declared Global Constant	Gas Optimization	Informational	✓
FM-05	Missing Check for Parameters	Optimization	Informational	✓
FM-06	Get Rewards In An Infinite Loop	Optimization	Major	✓
FM-07	Boolean Equality	Coding Style	Informational	✓
FM-08	Reentrancy Issue	Logical Issue	Informational	✓
FM-09	Permission for Function "claimRewards"	Permission	Minor	✓
FM-10	Unused Variables	Optimization	Informational	✓
XH-01	Missing Import Files	Optimization	Minor	✓
XH-02	Missing a Check for Creating ETH Stream	Optimization	Informational	✓

FM-01: Missing Emit Events

Type	Severity	Location
Optimization	Informational	FarmMaster.sol L174 L332

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like: `setVotingPool` `updatePool` in `FarmMaster.sol` .

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
    function setVotingPool(uint256 _pid) external onlyCore {
        votingPoolId = _pid;
        emit SetVotingPool(_pid);
    }
```

Alleviation:

The team had added the `emit event` in the functions.

The recommendations were applied in commit 1cebcef8fe22ed1fd136f00d76966359298e8bde.

FM-02: Missing Some Important Checks

Type	Severity	Location
Logical Issue	Informational	FarmMaster.sol L7 L48 XDEX.sol L20 Timelock.sol L75

Description:

Functions `setStream` and `setCore` in contract `FarmMaster.sol`, `setCore` in contract `XDEX.sol`, `constructor` in contract `Timelock.sol`.

All of them are missing address zero checks.

Recommendation:

Consider adding zero address check, for example:

```
    function setCore(address _core) public onlyCore {
        require(_core != address(0), "ERR_ZERO_ADDRESS");
        emit SET_CORE(core, _core);
        core = _core;
    }
```

Alleviation:

The team heeded our advice and added some checks for the functions.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-03: Proper Usage of "public" and "external" Type

Type	Severity	Location
Gas Optimization	Informational	FarmMaster.sol L192

Description:

`"public"` functions that are never called by the contract could be declared `"external"`. When the inputs are arrays `"external"` functions are more efficient than `"public"` functions.

Example:
Function `addPool` in contract `FarmMaster.sol`.

Recommendation:

Consider using the `"external"` attribute for functions never called from the contract.

Alleviation:

The team heeded our advice and changed the functions to `external`.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-04: State Variables That Could Be Declared Constant

Type	Severity	Location
Gas Optimization	Informational	FarmMaster.sol L357

Description:

Why is `lpReward` multiplied by `1e12` ? If necessary, please declare the `1e12` as a constant variable.

Recommendation:

Declare the `1e12` as a constant variable.

```
    uint256 private constant _lpRewardFix = 1e12;
```

Alleviation:

The team heeded our advice and declared the `1e12` as a constant variable.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-05: Missing Check for Parameters

Type	Severity	Location
Gas Optimization	Informational	FarmMaster.sol L427

Description:

The parameter `user` may not be available in function `deposit`, so it is recommended to add `require` before this.

```
    UserInfo storage user =
        poolInfo[_pid].LpTokenInfos[index].userInfo[msg.sender];
```

Recommendation:

Consider adding below checks before declaring variable.

```
    require(index < poolInfo[_pid].LpTokenInfos.length, "......");
    UserInfo storage user =
        poolInfo[_pid].LpTokenInfos[index].userInfo[msg.sender];
```

Alleviation:

The team heeded our advice and added the checks for parameters.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-06: Get Rewards In An Infinite Loop

Type	Severity	Location
Optimization	Major	FarmMaster.sol L490

Description:

The external function `deposit` can be called by the other contract. When the input parameter `_amount` is zero, the user can get unlimited initial deposit reward 10 XDEX.

```
    function deposit(
        uint256 _pid,
        IERC20 _lpToken,
        uint256 _amount
    ) external poolExists(_pid) {
        ......
        if (user.amount > 0) {
            ......
        } else {
            ......
            xdex.mint(address(this), bonusFirstDeposit);
            xdex.approve(address(stream), bonusFirstDeposit);
            stream.createStream(
                msg.sender,
                bonusFirstDeposit,
                StreamTypeVoting,
                streamStart
            );
            ......
        }
        ......
    }
```

Recommendation:

Add belowing checks in the first line of `deposit` unction.

```
    function deposit(
        uint256 _pid,
        IERC20 _lpToken,
        uint256 _amount
    ) external poolExists(_pid) {
        require(_amount > 0, "......");
        ......
    }
```

Alleviation:

The team heeded our advice and added the `require` check for the `_amount` parameter.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-07: Boolean Equality

Type	Severity	Location
Coding Style	Informational	FarmMaster.sol L491 L502

Description:

The variables `hasVotingStream` and `hasNormalStream` are boolean, we can use them directly as an expression.

```
    if (hasVotingStream == false) {
    ......
    if (hasNormalStream == false) {
```

Recommendation:

Consider using boolean variable directly.

```
    if (!hasVotingStream) {
    ......
    if (!hasNormalStream) {
```

Alleviation:

The team heeded our advice and changed the code.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-08: Reentrancy Issue

Type	Severity	Location
Logical Issue	Informational	FarmMaster.sol L611~L620

Description:

The function `emergencyWithdraw` could be called repeatedly, before the first invocation of the function was finished. This may cause the different invocations of the function to interact in destructive ways.

Since the user's balance is not set to 0, the second (and later) invocations will still succeed, and will withdraw the balance over and over again.

```
    lpInfo.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(
        msg.sender,
        _pid,
        address(lpInfo.lpToken),
        user.amount
    );
    user.amount = 0;
    user.rewardDebt = 0;
```

Recommendation:

Consider declaring a temporary variable to store `user.amount` . Clear `user.amount` before `lpInfo.lpToken.safeTransfer` . Sample is as below:

```
    uint256 amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
    lpInfo.lpToken.safeTransfer(address(msg.sender), amount);
    emit EmergencyWithdraw(
        msg.sender,
        _pid,
        address(lpInfo.lpToken),
        amount
    );
```

Alleviation:

The team heeded our advice and added the `nonReentrant` for the `emergencyWithdraw` function.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

FM-09: Permission for Function "claimRewards"

Type	Severity	Location
Permission	Minor	FarmMaster.sol L747

Description:

When does the core call function `claimRewards` ? This operation should be reflected in the white paper.

Recommendation:

Consider adding community voting for this function. And declare usage in the white page.

Alleviation:

The team will transfer the owner of this contract to `Timelock` .

The recommendations were applied in commit cbeafcf92936f1448aa7a161f3a07f82cf2c9d4c.

FM-10: Unused Variables

Type	Severity	Location
Optimization	Informational	FarmMaster.sol L16

Description:

The declared variables are not used, such as `onePercent` in `FarmMaster.sol` and `ONE` in `XdexStream`.

Recommendation:

Consider removing both of the unused variables.

Alleviation:

The team heeded our advice and removed the unused variable.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

XH-01: Missing Import Files

Type	Severity	Location
Optimization	Minor	XHalfLife.sol L6 L7

Description:

The following import files are not found in the audit files. And the relevant methods could not be verified.

```
import "./lib/AddressHelper.sol";
import "./lib/XNum.sol";
```

Recommendation:

Consider adding both of the import files.

Alleviation:

The team heeded our advice and added the files.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

XH-02: Missing a Check for Creating ETH Stream

Type	Severity	Location
Optimization	Informational	XHalfLife.sol L207

Description:

When the user creates an ETH stream, the input value should be greater than 0.0001ETH.

Recommendation:

Consider adding a `require` check in the function `createEtherStream`.

```
    function createEtherStream(
        address recipient,
        uint256 startBlock,
        uint256 kBlock,
        uint256 unlockRatio
    )
        external
        payable
        createStreamPreflight(recipient, msg.value, startBlock, kBlock)
        returns (uint256 streamId)
    {
        require(msg.value > 10**14, "deposit too small");
```

Alleviation:

The team heeded our advice and added check for creating ETH stream.

The recommendations were applied in commit 825808a96e650daeaa8b4817140f5ddea67f2e65.

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 instorage 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.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.