yAxis contest
Findings & Analysis Report

2022-01-27

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of yAxis contest smart contract system written in Solidity. The code contest took place between November 16—November 18 2021.

Wardens

18 Wardens contributed reports to the yAxis contest:

1. WatchPug (jtp and ming)
2. harleythedog
3. cmichel
4. pauliax
5. TimmyToes
6. defsec
7. ye0lde
8. hickuphh3
9. jonah1005
10. pmerkleplant
11. 0x0x0x
12. gzeon
13. tqts
14. hubble (ksk2345 and shri4net)

15. pants
16. xxxxx
17. hack3r-0m
18. Meta0xNull

This contest was judged by 0xleastwood.

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 15 unique vulnerabilities and 67 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity, 1 received a risk rating in the category of MEDIUM severity, and 12 received a risk rating in the category of LOW severity.

C4 analysis also identified 19 non-critical recommendations and 33 gas optimizations.

Scope

The code under review can be found within the C4 yAxis contest repository, and is composed of 184 smart contracts written in the Solidity programming language and includes 9265 lines of Solidity code and 7712 lines of JavaScript.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling

- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](#).

High Risk Findings (2)

[H-01] `YaxisVaultAdapter.sol#withdraw()` will most certainly fail

Submitted by WatchPug

The actual token withdrawn from `vault.withdraw()` will most certainly less than the `_amount`, due to precision loss in `_tokensToShares()` and `vault.withdraw()`.

As a result, `IDetailedERC20(_token).safeTransfer(_recipient, _amount)` will revert due to insufficant balance.

Based on the simulation we ran, it will fail `99.99%` of the time unless the `pps == 1e18`.

https://github.com/code-423n4/2021-11-yaxis/blob/146febcb61ae7fe20b0920849c4f4bbe111c6ba7/contracts/v3/alchemix/adapters/YaxisVaultAdapter.sol#L68-L72

```
function withdraw(address _recipient, uint256 _amount) external
    vault.withdraw(_tokensToShares(_amount));
    address _token = vault.getToken();
    IDetailedERC20(_token).safeTransfer(_recipient, _amount);
}
```

https://github.com/code-423n4/2021-11-yaxis/blob/146febcb61ae7fe20b0920849c4f4bbe111c6ba7/contracts/v3/Vault.sol#L181-L187

```
    function withdraw(
        uint256 _shares
    )
        public
        override
    {
        uint256 _amount = (balance().mul(_shares)).div(IERC20(addres
```

Recommendation

Change to:

```
    function withdraw(address _recipient, uint256 _amount) external
        address _token = vault.getToken();
        uint256 beforeBalance = IDetailedERC20(_token).balanceOf(adc

        vault.withdraw(_tokensToShares(_amount));

        IDetailedERC20(_token).safeTransfer(
            _recipient,
            IDetailedERC20(_token).balanceOf(address(this)) - before
        );
    }
```

Xuefeng-Zhu (yAxis) confirmed

[H-02] CDP.sol update overwrites user's credit on every positive increment

Submitted by harleythedog

Impact

Within `CDP.sol` (https://github.com/code-423n4/2021-11-yaxis/blob/main/contracts/v3/alchemix/libraries/alchemist/CDP.sol) there is a function called update. This function slowly decreases the debt of a position as yield is earned, until the debt is fully paid off, and the idea is then that the credit should begin incrementing as more yield is accumulated. However, the current logic to increment the totalCredit is this line of code (line 39 of `CDP.sol`):

```
\_self.totalCredit = \_earnedYield.sub(\_currentTotalDebt);
```

Notice that that each time update is called, this overwrites the previous totalCredit with the incremental credit accumulated. The line should instead be:

```
\_self.totalCredit =
\_self.totalCredit.add(\_earnedYield.sub(\_currentTotalDebt));
```

Indeed, look at the function `getUpdatedTotalCredit`, it returns the value:

```
\_self.totalCredit + (\_unclaimedYield - \_currentTotalDebt);
```

So it is obviously intended that the `totalCredit` should keep increasing over time instead of being overwritten on each update with a small value. The impact of this issue is large - the credit of every position will always be overwritten and the correct information will be lost forever. User's credit should grow over time, but instead it is overwritten with a small value every time update is called.

Proof of Concept

See line 39 in `CDP.sol` here: https://github.com/code-423n4/2021-11-yaxis/blob/main/contracts/v3/alchemix/libraries/alchemist/CDP.sol#:~:text=_self.totalCredit%20%3D%20_earnedYield.sub(_currentTotalDebt)%3B

Tools Used

Manual inspection.

Recommended Mitigation Steps

Change code as described above to increment `totalCredit` instead of overwrite it.

[Xuefeng-Zhu (yAxis) disputed]():

> If there is debt, the credit should be zero

0xleastwood (judge) commented:

> It seems like if `_self.totalDebt` is already zero and yield has been earned by the protocol, `_self.totalCredit` will be overwritten. This doesn't seem ideal, could you clarify why the issue is incorrect?

0xleastwood (judge) commented:

> If I'm not mistaken, yield can be earned from a positive credit (net 0 debt) position.

Xuefeng-Zhu (yAxis) commented:

> @0xleastwood `totalCredit` is 0 if there is debt

0xleastwood (judge) commented:

> After chatting to @Xuefeng-Zhu in Discord, he was able to confirm the issue as valid. So keeping it as is.

Medium Risk Findings (1)

[M-01] Prevent Minting During Emergency Exit

Submitted by TimmyToes, also found by defsec

Impact

Potential increased financial loss during security incident.

Proof of Concept

https://github.com/code-423n4/2021-11-yaxis/blob/0311dd421fb78f4f174aca034e8239d1e80075fe/contracts/v3/alchemix/Alchemist.sol#L611

Consider a critical incident where a vault is being drained or in danger of being drained due to a vulnerability within the vault or its strategies.

At this stage, you want to trigger emergency exit and users want to withdraw their funds and repay/liquidate to enable the withdrawal of funds. However, minting against debt does not seem like a desirable behaviour at this time. It only seems to enable unaware users to get themselves into trouble by locking up their funds, or allow an attacker to do more damage.

Recommended Mitigation Steps

Convert emergency exit check to a modifier, award wardens who made that suggestion, and then apply that modifier here.

Alternatively, it is possible that the team might want to allow minting against credit: users minting against credit would effectively be cashing out their rewards. This might be seen as desirable during emergency exit, or it might be seen as a potential extra source of risk. If this is desired, then the emergency exit check could be placed at line 624 with a modified message, instructing users to only use credit.

Xuefeng-Zhu (yAxis) confirmed

Low Risk Findings (12)

- [L-01] Pending governance is not cleared *Submitted by cmichel*
- [L-02] `Alchemist.sol` does not use safeApprove *Submitted by jonah1005, also found by cmichel*
- [L-03] `Alchemist.migrate` can push duplicate adapters to `_vaults` *Submitted by cmichel*
- [L-04] `Transmuter.unstake` updates user without first updating distributing yield *Submitted by cmichel*
- [L-05] No incentive to call `transmute()` instead of `forceTransmute(self)` *Submitted by cmichel*
- [L-06] anyone can deposit to adapters directly *Submitted by pauliax, also found by hickuphh3*
- [L-07] _setupRole not in constructor *Submitted by pauliax*

- [L-08] setSentinel actually adds sentinel *Submitted by pauliax*
- [L-09] Incorrect function docs *Submitted by pmerkleplant, also found by pauliax*
- [L-10] Incorrect Event Emitted in Alchemist.sol *Submitted by TimmyToes, also found by hubble, yeOlde, defsec, WatchPug, pauliax, and gzeon*
- [L-11] Incorrect comment or code in runPhasedDistribution (Transmuter.sol) *Submitted by yeOlde*
- [L-12] Effects and Interactions Before Check *Submitted by TimmyToes*

Non-Critical Findings (19)

- [N-01] Context and msg.sender *Submitted by pauliax*
- [N-02] Convert Emergency Exit Check to Modifier. *Submitted by TimmyToes*
- [N-04] Incorrect Info in Comment in Alchemist.sol *Submitted by TimmyToes*
- [N-05] Incorrect Info in Comment in Alchemist.sol (138) *Submitted by TimmyToes, also found by WatchPug*
- [N-06] Incorrect Comment *Submitted by TimmyToes*
- [N-07] Lack of 'emit' keyword in AlToken.sol *Submitted by tqts*
- [N-08] Use of deprecated `safeApprove` *Submitted by WatchPug, also found by pants and gzeon*
- [N-09] Should `safeApprove(0)` first *Submitted by WatchPug, also found by jonah1005 and defsec*
- [N-10] Lack of Proper Tests? *Submitted by TimmyToes*
- [N-11] Open TODOs *Submitted by pants*
- [N-12] Missing events for owner only functions that change critical parameters *Submitted by defsec, also found by WatchPug*
- [N-13] Require statements without messages *Submitted by pants*
- [N-14] No Transfer Ownership Pattern in AlToken.sol *Submitted by MetaOxNull*
- [N-15] Constructor Lack of Zero Address Check for Tokens *Submitted by MetaOxNull*
- [N-16] Tokens with fee on transfer are not supported *Submitted by WatchPug, also found by TimmyToes and 0x0x0x*

- [N-17] No Event Emitted on Minting *Submitted by TimmyToes*
- [N-18] Lack of Input Validation *Submitted by TimmyToes*
- [N-19] No event for `Alchemist.sol#setPegMinimum` *Submitted by 0x0x0x*
- [N-20] admin Variable is High Risk *Submitted by TimmyToes*

Gas Optimizations (33)

- [G-01] `AlchemistVault.sol` can be optimised *Submitted by 0x0x0x*
- [G-02] Optimize `Alchemist.sol#_withdrawFundsTo` *Submitted by 0x0x0x*
- [G-03] Cache length of array when looping *Submitted by 0x0x0x*
- [G-04] For uint `> 0` can be replaced with `!= 0` for gas optimisation *Submitted by 0x0x0x*
- [G-05] At `Alchemist.sol#acceptGovernance` , cache `pendingGovernance` earlier to save gas *Submitted by 0x0x0x, also found by xxxxx*
- [G-06] `CDP.sol#update.sol` can be optimized *Submitted by 0x0x0x*
- [G-07] `CDP.sol#getUpdatedTotalDebt` can be optimized *Submitted by 0x0x0x*
- [G-08] Upgrade pragma to at least 0.8.4 *Submitted by defsec*
- [G-09] Redundant Import *Submitted by defsec, also found by WatchPug*
- [G-10] Gas optimization: Caching variables *Submitted by gzeon*
- [G-11] Gas Optimization: Inline instead of modifier *Submitted by gzeon*
- [G-12] Gas optimization: Reduce storage write *Submitted by gzeon*
- [G-13] several functions can be marked external *Submitted by hack3r-0m, also found by defsec*
- [G-14] Remove FixedPointMath *Submitted by TimmyToes, also found by hickuphh3*
- [G-15] State variables can be `immutable` s *Submitted by pants*
- [G-16] Constant expressions *Submitted by pauliax*
- [G-17] Assigned operations to constant variables *Submitted by pauliax*
- [G-18] Multiple Assignments to Storage Variable *Submitted by TimmyToes*

- [G-19] Gas optimization when a paused user calls mint() in AlToken.sol *Submitted by tqts, also found by MetaOxNull and WatchPug*
- [G-20] Gas optimization in AlToken.sol *Submitted by tqts*
- [G-21] Unnecessary libraries *Submitted by WatchPug*
- [G-22] Use immutable variable can save gas *Submitted by WatchPug, also found by hickuphh3, pauliax, 0x0x0x, and TimmyToes*
- [G-23] Only using `SafeMath` when necessary can save gas *Submitted by WatchPug*
- [G-24] Cache and read storage variables from the stack can save gas *Submitted by WatchPug*
- [G-25] `YaxisVaultAdapter.sol` Use inline expression can save gas *Submitted by WatchPug*
- [G-26] Use short reason strings can save gas *Submitted by WatchPug, also found by pauliax*
- [G-27] Change unnecessary storage variables to constants can save gas *Submitted by WatchPug*
- [G-28] Save `vault.getToken()` as an immutable variable in `YaxisVaultAdapter.sol` contract can save gas *Submitted by WatchPug*
- [G-29] `Alchemist.sol#mint()` Two storage writes can be combined into one *Submitted by WatchPug*
- [G-30] Inline internal functions that are being used only once can save gas *Submitted by WatchPug*
- [G-31] Removing the unnecessary function *Submitted by xxxxx, also found by TimmyToes, pauliax, and hack3r-0m*
- [G-32] Unused Named Returns *Submitted by yeOlde*
- [G-33] TRANSMUTATION_PERIOD Issues *Submitted by yeOlde*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higherrisk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top