Wild Credit contest
Findings & Analysis Report

2021-11-16

Table of contents

Overview

About C4

Code 432n4 (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 Wild Credit contest smart contract system written in Solidity. The code contest took place between September 23—September 29 2021.

Wardens

15 Wardens contributed reports to the Wild Credit contest code contest:

- 0xRajeev
- WatchPug
- cmichel
- leastwood
- GalloDaSballo
- itsmeSTYJ
- gpersoon
- pauliax
- ye0lde
- hickuphh3
- tabish

- t11s
- jah
- pants

This contest was judged by ghoul.sol.

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 14 unique vulnerabilities and 63 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, 3 received a risk rating in the category of MEDIUM severity, and 9 received a risk rating in the category of LOW severity.

C4 analysis also identified 27 non-critical recommendations and 22 gas optimizations.

Scope

The code under review can be found within the C4 Wild Credit contest repository, and is composed of 40 smart contracts written in the Solidity programming language and includes 2,425 lines of Solidity code and 0 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 (3)

[H-01] Use of tokenB's price instead of tokenA in determining account health will lead to protocol mis-accounting and insolvency

Submitted by 0xRajeev, also found by WatchPug.

Impact

In `_supplyCreditUni()`, the last argument of `_convertTokenValues()` on `L674` being `_priceB` instead of `_priceA` in the calculation of `supplyB` is a typo (should be `_priceA`) and therefore miscalculates `supplyB`, `creditB`, `creditUni` and therefore `totalAccountSupply` in function `accountHealth()` which affects the health of account/protocol determination that is used across all borrows/withdrawals/transfers/liquidations in the protocol. This miscalculation significantly affects all calculations in protocol and could therefore cause protocol insolvency.

Proof of Concept

- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L674
- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L340
- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L398-L401

- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L532
- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L544
- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L119
- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L266
- https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/LendingPair.sol#L289

Tools Used

Manual Analysis

Recommended Mitigation Steps

Change the last argument of `\_convertTokenValues()` from `\_priceB` to `\_priceA` on L674.

talegift (Wild Credit) confirmed

[H-02] Liquidation can be escaped by depositing a Uni v3 position with 0 liquidity

Submitted by WatchPug.

When the liquidator is trying to liquidate a undercolldarezed loan by calling `liquidateAccount()`, it calls `_unwrapUniPosition()` -> `uniV3Helper.removeLiquidity()` -> `positionManager.decreaseLiquidity()`.

However, when the Uni v3 position has 0 liquidity, `positionManager.decreaseLiquidity()` will fail.

See: https://github.com/Uniswap/v3-periphery/blob/main/contracts/NonfungiblePositionManager.sol#L265

Based on this, a malicious user can escaped liquidation by depositing a Uni v3 position with 0 liquidity.

Impact

Undercollateralized debts cannot be liquidated and it leads to bad debts to the protocol.

A malicious user can take advantage of this by creating long positions on the collateral assets and take profit on the way up, and keep taking more debt out of the protocol, while when the price goes down, the debt can not be liquidated and the risks of bad debt are paid by the protocol.

Proof of Concept

1. A malicious user deposits some collateral assets and borrow the max amount of debt;
2. The user deposits a Uni v3 position with 0 liquidity;
3. When the market value of the collateral assets decreases, the liquadation will fail as `positionManager.decreaseLiquidity()` reverts.

Recommendation

Check if liquidity > 0 when removeLiquidity.

talegift (Wild Credit) confirmed:

> Valid issue. Good catch.

> Severity should be lowered to 2 as it doesn't allow direct theft of funds and the loss would only occur under specific external conditions.

> *2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements*

> https://docs.code4rena.com/roles/wardens/judging-criteria#estimating-risk-tl-dr

ghoul-sol (judge) commented:

> To my understanding, bad position would affect the whole protocol and a loss would have to be paid by other participans which means funds can be drained. For that reason, I'm keeping high risk.

Medium Risk Findings (4)

[M-01] Use of deprecated Chainlink API

Submitted by 0xRajeev, also found by cmichel and leastwood.

Impact

The contract uses Chainlink's deprecated API `latestAnswer()`. Such functions might suddenly stop working if Chainlink stopped supporting deprecated APIs.

Impact: Deprecated API stops working. Prices cannot be obtained. Protocol stops and contracts have to be redeployed.

See similar Low-severity finding L11 from OpenZeppelin's Audit of Opyn Gamma Protocol: https://blog.openzeppelin.com/opyn-gamma-protocol-audit/

This was a Medium-severity finding even in the previous version of WildCredit contest as well: https://github.com/code-423n4/2021-07-wildcredit-findings/issues/75 where it was reported that " `latestAnswer` method will return the last value, but you won't be able to check if the data is fresh. On the other hand, calling the method `latestRoundData` allows you to run some extra validations."

Proof of Concept

https://github.com/code-423n4/2021-09-wildcredit/blob/c48235289a25b2134bb16530185483e8c85507f8/contracts/UniswapV3Oracle.sol#L101

See https://docs.chain.link/docs/deprecated-aggregatorinterface-api-reference/#latestanswer.

Tools Used

Manual Analysis

Recommended Mitigation Steps

Use V3 interface functions: https://docs.chain.link/docs/price-feeds-api-reference/

talegift (Wild Credit) acknowledged::

> We'll remove dependence on Chainlink completely.

[M-02] `LendingPair.withdrawUniPosition` should accrue debt first

Submitted by cmichel.

The `LendingPair.withdrawUniPosition` function allows the user to withdraw their UniswapV3 pool position (NFT) again. As the Uniswap position acts as collateral in the protocol, a health check is performed afterwards.

However, it does not check the **current** debt of the caller as it does not `accrue` the debt for both tokens first.

Impact

In the worst case, in low-activity markets, it could happen that debt has not accrued for a long time and the current debt is significantly higher than the current *recorded* debt in `totalDebtAmount`. An account with a de-facto negative health ratio if the debt was accrued could still withdraw their collateral NFT instead of having to repay their debt first.

Recommendation

Accrue the debt for both tokens first in `LendingPair.withdrawUniPosition`.

talegift (Wild Credit) confirmed

[M-03] Supply part of the accrued debt can be stolen

Submitted by cmichel.

The `LendingPair.uniClaimDeposit` function allows the user to "collect fees" and mint new supply shares with the collected amounts.

`uniClaimDeposit` does not accrue tokens

However, the current total supply is not `accrue` d in the function. This means an attacker can:

- mint shares using `uniClaimDeposit`
- increase the `totalSupplyAmount` by calling `accrue(token0)` and `accrue(token1)` afterwards.
- call `withdraw` and receive *a larger amount of tokens* for the newly minted shares due to the increase in `totalSupplyAmount` from `accrue` (increasing the supply share price `_sharesToSupply`).

This would only lead to a small protocol loss if `uniClaimDeposit` would only collect the *fees*, however, combined with another flaw, one can steal almost the entire protocol `lpRate` each time:

`uniClaimDeposit` allows collecting entire liquidity instead of just fees

This has to do with the way liquidity from a Uniswap V3 position (NFT) is withdrawn:

- When calling `positionManager.decreaseLiquidity`, the `position.liquidity` is removed but stored in the position as `tokensOwed0/tokensOwed1`. It is **not** transferred to the user.
- One needs to call `positionManager.collect(params)` to actually transfer out these tokens, setting `tokensOwed0/1` to `0`. (This is correctly done in `UniswapV3Helper.removeLiquidity`.)

An attacker can perform the following attack:

- Create a Uniswap V3 position.
- Get flashloans for both tokens to provide lots of liquidity for this position.
- Call `positionManager.decreaseLiquidity` such that the entire liquidity is removed and stored (but not collected yet) in the position's `tokensOwed0/1` fields
- Deposit it to WildCredit's lending pair using `depositUniPosition`
- Call `uniClaimDeposit` to mint a huge amount of NFT supply shares. This huge amount will capture the protocol's debt accrual in the next steps.
- Call `accrue` on both tokens to accrue debt and pay the `lpRate` part of it to suppliers, increasing `totalSupplyAmount` and thus the value of a supply share.
- With the new debt added to the `totalSupplyAmount`, the attacker can now withdraw their minted shares again and capture most of the new debt that was accrued, making a profit.

Impact

Combining these two issues, an attacker could steal most of the accrued `lpRate` in a single atomic transaction. The attacker can repeat this step capturing the supplier interest for each accrual. (The longer the market hasn't been accrued, the bigger the profit per single attack transaction, but in the end, the attacker could perform this attack at every block or when it becomes profitable for the gas costs.)

Providing / removing Uniswap V3 liquidity does not incur fees.

The attacker's profit is the loss of other legitimate suppliers that capture less of the newly accrued debt.

Recommendation

Accrue the debt for both tokens first in `LendingPair.uniClaimDeposit`.

It might also be a good idea to disallow collecting the "parked" liquidity in a token (that has been removed but not yet collected) by immediately collecting them when the NFT is deposited in `depositUniPosition`. I.e., call `_uniCollectFees` in `depositUniPosition` to withdraw any outstanding tokens and fees. Then mint shares with these token amounts.

talegift (Wild Credit) confirmed:

> We'll implement the suggested fix.

> Suggest lowering severity to 2 as it doesn't allow direct theft of funds and the loss would only occur under specific external conditions - long periods of not accrue interest combined with a low gas price to steal the pending interest.

ghoul-sol (judge) commented:

> It seems that the attacker can steal interest that is owed to other users but deposits are safe. For that reason I agree with sponsor to make this medium risk.

Low Risk Findings (10)

- [L-01] Missing SafeMath *Submitted by 0xRajeev.*
- [L-02] Constraint of minRate < lowRate can be broken *Submitted by 0xRajeev, also found by GalloDaSballo and itsmeSTYJ.*
- [L-03] Missing threshold check for highRate *Submitted by 0xRajeev, also found by itsmeSTYJ.*
- [L-04] Uniswap oracle assumes PairToken <> WETH liquidity *Submitted by cmichel.*
- [L-05] Simple interest formula is used *Submitted by cmichel.*
- [L-06] Reduce risk of rounding error in _timeRateToBlockRate *Submitted by gpersoon, also found by pauliax.*
- [L-07] Race condition on ERC20 approval *Submitted by itsmeSTYJ.*
- [L-08] Oracle response assumes 8 decimals *Submitted by pauliax.*
- [L-09] Oracle should call latestRoundData instead.

Non-Critical Findings (28)

- [N-01] Missing event for this critical onlyOperator function where the operator can arbitrarily change name+symbol *Submitted by 0xRajeev.*
- [N-02] Missing zero-address checks *Submitted by 0xRajeev, also found by GalloDaSballo and yeOlde.*

- [N-03] Strict inequality should be relaxed to be closed ranges instead of open *Submitted by 0xRajeev.*
- [N-04] Incorrect error message strings with require()s *Submitted by 0xRajeev, also found by WatchPug.*
- [N-05] Remove pair-specific parameters until they are actually used/enforced *Submitted by 0xRajeev.*
- [N-06] Using a zero-address check as a proxy for enforcing one-time initialization is risky *Submitted by 0xRajeev.*
- [N-07] Renouncing ownership is not allowed *Submitted by 0xRajeev.*
- [N-08] Lack of guarded launch approach may be risky *Submitted by 0xRajeev.*
- [N-09] Clone-and-own approach used for OZ libraries is susceptible to errors and missing upstream bug fixes *Submitted by 0xRajeev.*
- [N-10] Lack of check for address(0) in `LendingPair.depositUniPosition` *Submitted by GalloDaSballo.*
- [N-11] Missing parameter validation *Submitted by cmichel.*
- [N-12] `setTargetUtilization()` Misleading error message *Submitted by WatchPug, also found by cmichel, gpersoon, pauliax, and itsmeSTYJ.*
- [N-13] Truncated math in `interestRatePerBlock` *Submitted by cmichel.*
- [N-14] `UniswapV3Helper.getUserTokenAmount` could be simplified *Submitted by cmichel.*
- [N-15] Add nonReentrant modifiers to uniswap position methods + Check effects pattern *Also found by gpersoon.*
- [N-16] UniswapV3Helper: Misleading param names for getSqrtPriceX96() *Submitted by hickuphh3.*
- [N-17] Only accept ETH from WETH contract *Submitted by pauliax.*
- [N-18] Ensure targetUtilization > 0 *Submitted by pauliax.*
- [N-19] Incorrect import *Submitted by tabish.*
- [N-20] transferLp() Misleading error message *Submitted by WatchPug.*
- [N-21] The check if _checkBorrowEnabled and _checkBorrowLimits can be done earlier *Submitted by WatchPug.*

- [N-22] Consider adding `account` parameter to event WithdrawUniPosition *Submitted by WatchPug.*
- [N-23] Improve readability of constants *Submitted by gpersoon.*
- [N-24] Improper File Imports *Submitted by leastwood.*
- [N-25] Emit events when setting the initial values in the constructor *Submitted by pauliax.*
- [N-26] Style issues *Submitted by pauliax.*
- [N-27] Prefer abi.encode over abi.encodePacked *Submitted by t11s.*

Gas Optimizations (22)

- [G-01] Caching state variables in local/memory variables avoids SLOADs to save gas *Submitted by 0xRajeev.*
- [G-02] Redundant zero-address checks *Submitted by 0xRajeev.*
- [G-03] Input validation on positionID not being 0 will save gas *Submitted by 0xRajeev.*
- [G-04] Input validation on amount > 0 will save gas *Submitted by 0xRajeev, also found by WatchPug.*
- [G-05] Use unchecked{} primitive to save gas where possible *Submitted by 0xRajeev.*
- [G-06] Moving checks before other logic can save gas *Submitted by 0xRajeev.*
- [G-07] Unused parameter removal can save gas *Submitted by 0xRajeev, also found by yeOlde.*
- [G-08] Using msg.sender or cached locals in emits instead of state variables saves gas *Submitted by 0xRajeev.*
- [G-09] Avoiding unnecessary SSTORE can save gas *Submitted by 0xRajeev.*
- [G-10] Reordering state variable declarations to prevent incorrect packing can save slots/gas *Submitted by 0xRajeev.*
- [G-11] Making PairFactory state vars immutable would save gas *Submitted by t11s, also found by 0xRajeev and jah.*
- [G-12] Change unnecessary _borrowBalanceConverted to _debtOf can save gas *Submitted by WatchPug.*

- [G-13] Change unnecessary _supplyBalanceConverted to _supplyOf can save gas *Submitted by WatchPug.*
- [G-14] Cache and check decimals before write storage can save gas *Submitted by WatchPug.*
- [G-15] Gas: Unnecessary `_maxAmount` parameter in `repayAllETH` *Submitted by cmichel, also found by WatchPug.*
- [G-16] UniswapV3Helper: Avoid recomputation of sqrtRatio from pool tick *Submitted by hickuphh3.*
- [G-17] UniswapV3Helper: Redundant pool initialization *Submitted by hickuphh3.*
- [G-18] UniV3Helper: Function visibilities can be restricted to pure *Submitted by hickuphh3.*
- [G-19] Declare the value when the variable is created *Submitted by jah.*
- [G-20] PairFactory.sol is Ownable but not owner capabilites are used *Submitted by jah.*
- [G-21] Unused imports *Submitted by pauliax.*
- [G-22] Use unchecked{} in ERC20 to save gas without risk *Submitted by t11s.*

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 higher-risk 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