Yeti Finance contest
Findings & Analysis Report

2022-03-02

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 Yeti Finance contest smart contract system written in Solidity. The code contest took place between December 16—December 22 2021.

Wardens

27 Wardens contributed reports to the Yeti Finance contest:

1. kenzo
2. jayjonah8
3. cmichel
4. hyh
5. WatchPug (jtp and ming)
6. UncleGrandpa925
7. pauliax
8. dalgarim
9. csanuragjain
10. 0x1f8b
11. heiho1
12. Jujic
13. defsec
14. robee
15. Ruhum
16. gzeon
17. certora
18. Dravee
19. shenwilly
20. p4st13r4 (0xb4bb4 and 0x69e8)
21. gpersoon
22. SolidityScan
23. sirhashalot
24. cccz
25. broccolirob

This contest was judged by Alberto Cuesta Cañada.

Final report assembled by captainmango and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 55 unique vulnerabilities and 129 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 8 received a risk rating in the category of HIGH severity, 13 received a risk rating in the category of MEDIUM severity, and 33 received a risk rating in the category of LOW severity.

C4 analysis also identified 18 non-critical recommendations and 56 gas optimizations.

33 of the findings for this contest relate to a wrapped asset subsystem (and in particular, the `WJLP` contract) that was noted as experimental, and has subsequently been removed from the Yeti Finance protocol. Of these WJLP / wrapped asset findings, 6 were high risk and 4 were medium risk. At the time of the C4 audit contest launch, this subsystem was known to be not well tested and experimental, as noted in the contest repo; therefore the wardens were advised to consider it as an example only. Because this one contract had an outsized impact on the outcome of this contest, and since it has since been removed, we have organized the report into two sections, separating out the high- and medium-risk findings related to WJLP.

Scope

The code under review can be found within the C4 Yeti Finance contest repository, and is composed of 14 smart contracts and includes 4459 source lines of Solidity code.

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] receiveCollateral() can be called by anyone

Submitted by jayjonah8, also found by dalgarim and kenzo

Impact

In StabilityPool.sol, the receiveCollateral() function should be called by ActivePool per comments, but anyone can call it passing in _tokens and _amounts args to update stability pool balances.

Proof of Concept

https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/StabilityPool.sol#L1143

Recommended Mitigation Steps

Allow only the ActivePool to call the receiveCollateral() function: require(msg.sender = address(active pool address), “Can only be called by ActivePool”)

kingyetifinance(Yeti finance) confirmed:

> @LilYeti: This was also caught by our official auditor, but good catch.

0xtruco (Yeti finance) commented:

> Fixed this, #190, #285, already in code https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/StabilityPool.sol#L1144

[H-02] Yeti token rebase checks the additional token amount incorrectly

Submitted by hyh

Impact

The condition isn't checked now as the whole balance is used instead of the Yeti tokens bought back from the market. As it's not checked, the amount added to `effectiveYetiTokenBalance` during rebase can exceed the actual amount of the Yeti tokens owned by the contract. As the before check amount is calculated as the contract net worth, it can be fixed by immediate buy back, but it will not be the case.

The deficit of Yeti tokens can materialize in net worth terms as well if Yeti tokens price will raise compared to the last used one. In this case users will be cumulatively accounted with the amount of tokens that cannot be actually withdrawn from the contract, as its net holdings will be less then total users' claims. In other words, the contract will be in default if enough users claim after that.

Proof of Concept

Now the whole balance amount is used instead of the amount bought back from market.

Rebasing amount is added to `effectiveYetiTokenBalance`, so it should be limited by extra Yeti tokens, not the whole balance: https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/YETI/sYETIToken.sol#L247

Recommended Mitigation Steps

It looks like only extra tokens should be used for the check, i.e. `yetiToken.balance - effectiveYetiTokenBalance`.

Now:

```
function rebase() external {
        ...
    uint256 yetiTokenBalance = yetiToken.balanceOf(address(this)
    uint256 valueOfContract = _getValueOfContract(yetiTokenBalar
    uint256 additionalYetiTokenBalance = ...
    if (yetiTokenBalance < additionalYetiTokenBalance) {
            additionalYetiTokenBalance = yetiTokenBalance;
    }
    effectiveYetiTokenBalance = effectiveYetiTokenBalance.add(ac
...
function _getValueOfContract(uint _yetiTokenBalance) internal vi
    uint256 adjustedYetiTokenBalance = _yetiTokenBalance.sub(eff
    uint256 yusdTokenBalance = yusdToken.balanceOf(address(this)
    return div(lastBuybackPrice.mul(adjustedYetiTokenBalance),
}
```

As the `_getValueOfContract` function isn't used elsewhere, the logic can be simplified. To be:

```
function rebase() external {
    ...
    uint256 adjustedYetiTokenBalance = (yetiToken.balanceOf(addr
    uint256 valueOfContract = _getValueOfContract(adjustedYetiTc
    uint256 additionalYetiTokenBalance = ...
    if (additionalYetiTokenBalance > adjustedYetiTokenBalance) {
            additionalYetiTokenBalance = adjustedYetiTokenBalanc
    }
    effectiveYetiTokenBalance = effectiveYetiTokenBalance.add(ac
...
function _getValueOfContract(uint _adjustedYetiTokenBalance) int
    uint256 yusdTokenBalance = yusdToken.balanceOf(address(this)
    return div(lastBuybackPrice.mul(_adjustedYetiTokenBalance),
}
```

kingyetifinance (Yeti finance) disagreed with severity and confirmed:

> @LilYeti:

> This is the logic for the fix which we have already done:

> if (yetiTokenBalance - effectiveYetiTokenBalance < additionalYetiTokenBalance)

> Will look into this again before confirming as fixed to see if it is the same as the suggested error.

[Oxtruco (Yeti finance) commented]():

> https://github.com/code-423n4/2021-12-yetifinance/pull/12

Medium Risk Findings (09)

[M-01] Wrong `lastBuyBackPrice`

Submitted by cmichel

The `sYETIToken.lastBuyBackPrice` is set in `buyBack` and hardcoded as:

```
    function buyBack(address routerAddress, uint256 YUSDToSell, uint
        require(YUSDToSell > 0, "Zero amount");
        require(lastBuybackTime + 69 hours < block.timestamp, "Must
        yusdToken.approve(routerAddress, YUSDToSell);
        uint256[] memory amounts = IRouter(routerAddress).swapExact]
        lastBuybackTime = block.timestamp;
        // amounts[0] is the amount of YUSD that was sold, and amour
        // @audit this hardcoded lastBuybackPrice is wrong when usir
        lastBuybackPrice = div(amounts[0].mul(1e18), amounts[1]);
        emit BuyBackExecuted(YUSDToSell, amounts[0], amounts[1]);
    }
```

It divides the first and second return `amounts` of the swap, however, these amounts depend on the swap `path` parameter that is used by the caller. If a swap path of length 3 is used, then this is obviously wrong. It also assumes that each router sorts the pairs the same way (which is true for Uniswap/Sushiswap).

Impact

The `lastBuyBackPrice` will be wrong when using a different path. This will lead `rebase`s using a different yeti amount and the `effectiveYetiTokenBalance` being updated wrong.

Recommended Mitigation Steps

Verify the first and last element of the path are YETI/YUSD and use the first and last amount parameter.

kingyetifinance (Yeti finance) confirmed and disagreed with severity

> @LilYeti: The idea was that on launch we will likely use a curve pool to route through so this contract would change slightly. However it is valid and some more checks would be good to add. Moving to level 1 issue.

alcueca (Judge) commented:

> A medium severity rating is warranted.

[M-02] Should check return data from Chainlink aggregators

Submitted by defsec, also found by hyh and WatchPug

Impact

The latestRoundData function in the contract PriceFeed.sol fetches the asset price from a Chainlink aggregator using the latestRoundData function. However, there are no checks on roundID.

Stale prices could put funds at risk. According to Chainlink's documentation, This function does not error if no answer has been reached but returns 0, causing an incorrect price fed to the PriceOracle. The external Chainlink oracle, which provides index price information to the system, introduces risk inherent to any dependency on third-party data sources. For example, the oracle could fall behind or otherwise fail to be maintained, resulting in outdated data being fed to the index price calculations of the liquidity.

Example Medium Issue : https://github.com/code-423n4/2021-08-notional-findings/issues/18

Proof of Concept

1. Navigate to the following contract.

https://github.com/code-423n4/2021-12-yetifinance/blob/1da782328ce4067f9654c3594a34014b0329130a/packages/contracts/contracts/PriceFeed.sol#L578

2. Only the following checks are implemented.

```
if (!_response.success) {return true;}
// Check for an invalid roundId that is 0
if (_response.roundId == 0) {return true;}
// Check for an invalid timeStamp that is 0, or in the futur
if (_response.timestamp == 0 || _response.timestamp > block.
// Check for non-positive price
if (_response.answer <= 0) {return true;}
```

Recommended Mitigation Steps

Consider to add checks on the return data with proper revert messages if the price is stale or the round is incomplete, for example:

```
(uint80 roundID, int256 price, , uint256 timeStamp, uint80 answe
require(price > 0, "Chainlink price <= 0");
require(answeredInRound >= roundID, "...");
require(timeStamp != 0, "...");
```

kingyetifinance (Yeti finance) confirmed:

> @LilYeti:
>
> https://docs.chain.link/docs/faq/#how-can-i-check-if-the-answer-to-a-round-is-being-carried-over-from-a-previous-round
>
> https://github.com/code-423n4/2021-08-notional-findings/issues/92

[M-03] Unwhitelisted token can cause disaster

Submitted by csanuragjain

Impact

Contract instability and financial loss. This will happen if one of the allowed contract calls sendCollaterals with non whitelisted token (may happen with user input on allowed contract)

Proof of Concept

1. Navigate to contract at https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/ActivePool.sol
2. Assume sendCollaterals function is called by one of allowed contract with a non whitelisted token and amount as 1

```
function sendCollaterals(address _to, address[] memory _tokens,
    _requireCallerIsBOorTroveMorTMLorSP();
    require(_tokens.length == _amounts.length);
    for (uint i = 0; i < _tokens.length; i++) {
        _sendCollateral(_to, _tokens[i], _amounts[i]); // revert
    }

    if (_needsUpdateCollateral(_to)) {
        ICollateralReceiver(_to).receiveCollateral(_tokens, _amc
    }

    return true;
}
```

3. This calls _sendCollateral with our non whitelisted token and amount as 1

```
function _sendCollateral(address _to, address _collateral, uint
    uint index = whitelist.getIndex(_collateral);
    poolColl.amounts[index] = poolColl.amounts[index].sub(_amour
    bool sent = IERC20(_collateral).transfer(_to, _amount);
    require(sent);

    emit ActivePoolBalanceUpdated(_collateral, _amount);
    emit CollateralSent(_collateral, _to, _amount);
}
```

4. whitelist.getIndex(_collateral); will return 0 as our collateral is not whitelisted and will not be present in whitelist.getIndex(_collateral);. This means index will point to whitelisted collateral at index 0
5. poolColl.amounts[index] will get updated for whitelisted collateral at index 0 even though this collateral was never meant to be updated

```
poolColl.amounts[index] = poolColl.amounts[index].sub(_amount);
```

6. Finally our non supported token gets transferred to recipient and since _needsUpdateCollateral is true so recipient poolColl.amounts gets increased even though recipient never received the whitelisted collateral
7. Finally sender pool amount will be reduced even though it has the whitelisted collateral and recipient pool amount will be increased even though it does not have whitelisted collateral

Recommended Mitigation Steps

Add a check to see if collateral to be transferred is whitelisted

kingyetifinance (Yeti finance) disputed:

> @LilYeti: Thanks for the thorough run through. It is true, but this is abstracted away, all calls of sendCollateral are internal / between contracts in our codebase, and there are checks for valid collateral in whitelist before this.

alcueca (Judge) commented:

> Validating data integrity outside a function inside the same contract would be a low severity. Validating data integrity in an external contract is medium severity. Many things can go wrong.

[M-04] Out of gas.

Submitted by Jujic, also found by gzeon

There is no upper limit on `poolColl.tokens[]`, it increments each time when a new collateral is added. Eventually, as the count of collateral increases, gas cost of

smart contract calls will raise and that there is no implemented function to reduce the array size.

Impact

For every call `getVC()` function which computed contain the VC value of a given collateralAddress is listed in `poolColl.tokens[]` array, the gas consumption can be more expensive each time that a new collateral address is appended to the array, until reaching an “Out of Gas” error or a “Block Gas Limit” in the worst scenario.

Proof of Concept

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/ActivePool.sol#L268

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/DefaultPool.sol#L184

Tools Used

Remix

Recommended Mitigation Steps

Array’s length should be checked.

kingyetifinance (Yeti finance) confirmed and disagreed with severity:

> @LilYeti: This is a known problem, and we are yet to test the upper limits of the contracts as is. Not sure how more theoretical issues like these are scored, but I would agree with that it is a medium to high risk based on how likely it is to happen * the potential effects. The worst possible outcome is that funds are locked in the protocol because it costs too much gas to do a withdrawal. We are still doing analysis on this, judges do what you want with this information. We would actually recommend it be a severity level 2, but it does have high potential risk.

[M-05] Reentrancy in contracts/BorrowerOperations.sol

Submitted by heiho1, also found by jayjonah8

Impact

There are several potential re-entrant functions in contracts/BorrowerOperations.sol:

=> Function addColl() on line 346 is potentially re-entrant as it is external but has no re-entrancy guard declared. This function invokes _adjustTrove() which potentially impacts user debt, collateral top-ups or withdrawals.

- Same applies to

-- withdrawColl() on line 373 -- withdrawYUSD() on line 389 -- repayYUSD() on line 406 -- adjustTrove() on line 420

=> Function openTrove() on line 207 is potentially re-entrant as it is external but has no re-entrancy guard declared. This function invokes _openTroveInternal() which potentially impacts trove creation, YUSD withdrawals and YUSD gas compensation.

=> Function closeTrove() on line 628 is potentially re-entrant as it is external but has no re-entrancy guard declared. This function invokes troveManagerCached.removeStake(msg.sender) and troveManagerCached.closeTrove(msg.sender) impacting outcomes like debt, rewards and trove ownership.

Proof of Concept

https://solidity-by-example.org/hacks/re-entrancy/

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/BorrowerOperations.sol#L346

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/BorrowerOperations.sol#L373

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/BorrowerOperations.sol#L389

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/BorrowerOperations.sol#L420

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/BorrowerOperations.sol#L207

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/BorrowerOperations.sol#L628

Tools Used

Slither

Recommended Mitigation Steps

Potential solution is a re-entrancy guard similar to https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard

[M-06] Collateral parameters can be overwritten

Submitted by cmichel, also found by csanuragjain and gzeon

It's possible to repeatedly add the first collateral token in `validCollateral` through the `Whitelist.addCollateral` function. The `validCollateral[0] != _collateral` check will return false and skip further checks.

POC

Owner calls `addCollateral(collateral=validCollateral[0])`:

```
    function addCollateral(
        address _collateral,
```

```
        uint256 _minRatio,
        address _oracle,
        uint256 _decimals,
        address _priceCurve,
        bool _isWrapped
    ) external onlyOwner {
        checkContract(_collateral);
        checkContract(_oracle);
        checkContract(_priceCurve);
        // If collateral list is not 0, and if the 0th index is not
        // then if index is 0 that means it is not set yet.
        // @audit evaluates validCollateral[0] != validCollateral[0]
        if (validCollateral.length != 0 && validCollateral[0] != _cc
            require(collateralParams[_collateral].index == 0, "colla
        }

        validCollateral.push(_collateral);
        // overwrites parameters
        collateralParams[_collateral] = CollateralParams(
            _minRatio,
            _oracle,
            _decimals,
            true,
            _priceCurve,
            validCollateral.length - 1,
            _isWrapped
        );
    }
```

Impact

The collateral parameters `collateralParams` are re-initialized which can break the existing accounting. The collateral token also exists multiple times in `validCollateral`.

Recommended Mitigation Steps

Fix the check. It should be something like:

```
    if (validCollateral.length > 0) {
        require(collateralParams[_collateral].index == 0 && validCol
    }
```

[M-07] Cannot use most piecewise linear functions with current implementation

Submitted by cmichel

The `ThreePieceWiseLinearPriceCurve.adjustParams` function uses three functions `f1, f2, f3` where `y_i = f_i(x_i)`. It computes the y-axis intersect (`b2 = f_2(0), b3 = f_3(0)`) for each of these but uses **unsigned integers** for this, which means these values cannot become negative. This rules out a whole class of functions, usually the ones that are desirable.

Example:

Check out this two-piece linear interest curve of Aave:

The intersection of the second steep straight line with the y-axis `b_2 = f_2(0)` would be negative.

Example: Imagine a curve that is flat at `10%` on the first 50% utilization but shoots up to `110%` at 100% utilization.

- `m1 = 0, b1 = 10%, cutoff1 = 50%`
- `m2 = 200%` => `b2 = m1 * cutoff1 + b1 - m2 * cutoff1 = f1(cutoff1) - m2 * cutoff1 = 10% - 200% * 50% = 10% - 100% = -90%`. (`f2(100%) = 200% * 100% - 90% = 110%` ✅)

This function would revert in the `b2` computation as it underflows due to being a negative value.

Impact

Most curves that are actually desired for a lending platform (becoming steeper at higher utilization) cannot be used.

Recommended Mitigation Steps

Evaluate the piecewise linear function in a different way that does not require computing the y-axis intersection value. For example, for `cutoff2 >= x > cutoff1`, use `f(x) = f_1(cutoff) + f_2(x - cutoff)`. See Compound.

kingyetifinance (Yeti finance) confirmed:

> @LilYeti: Great find.

Oxtruco (Yeti finance) commented:

> Resolved in https://github.com/code-423n4/2021-12-yetifinance/pull/23 by adding negative possibility

[M-08] Wrong comment in `getFee`

Submitted by cmichel

The `ThreePieceWiseLinearPriceCurve.getFee` comment states that the total + the input must be less than the cap:

> If dollarCap == 0, then it is not capped. Otherwise, **then the total + the total input** must be less than the cap.

The code only checks if the input is less than the cap:

```
// @param _collateralVCInput is how much collateral is being inp
if (dollarCap != 0) {
    require(_collateralVCInput <= dollarCap, "Collateral input e
}
```

Recommended Mitigation Steps

Clarify the desired behavior and reconcile the code with the comments.

kingyetifinance (Yeti finance) confirmed and disagreed with severity:

> @LilYeti: This was an issue also found by one of our independent economic auditors. Good find. Is actually more like a medium (severity 2) issue.

Oxtruco (Yeti finance) commented:

> Fixed in line 92

[M-09] Fee not decayed if past `decayTime`

Submitted by cmichel

The `ThreePieceWiseLinearPriceCurve.calculateDecayedFee` function is supposed to decay the `lastFeePercent` over time. This is correctly done in the `decay > 0 && decay < decayTime` case, but for the `decay > decayTime` case it does not decay at all but should set it to 0 instead..

```
if (decay > 0 && decay < decayTime) {
    // @audit if decay is close to decayTime, this fee will be z
    fee = lastFeePercent.sub(lastFeePercent.mul(decay).div(decay
} else {
    fee = lastFeePercent;
}
```

Recommended Mitigation Steps

It seems wrong to handle the `decay == 0` case (decay happened in same block) the same way as the `decay >= decayTime` case (decay happened long time ago) as is done in the `else` branch. I believe it should be like this instead:

```
// decay == 0 case should be full lastFeePercent
if(decay < decayTime) {
    fee = lastFeePercent.sub(lastFeePercent.mul(decay).div(decay
} else {
    // reset to zero if decay >= decayTime
    fee = 0;
}
```

kingyetifinance (Yeti finance) confirmed:

> @LilYeti: Good find. The fee would be reset to not 0 in this case.

0xtruco (Yeti finance) commented:

> Resolved in https://github.com/code-423n4/2021-12-yetifinance/pull/14.

Low Risk Findings (33)

- [L-01] Must approve 0 first *Submitted by robee*
- [L-02] TellorCaller.sol constructor does not guard against zero address *Submitted by jayjonah8*
- [L-03] Unipool's and Pool2Unipool's setParams can be run repeatedly *Submitted by hyh*
- [L-04] Use safeTransfer/safeTransferFrom consistently instead of transfer/transferFrom *Submitted by defsec, also found by 0x1f8b, broccolirob, certora, cmichel, csanuragjain, hyh, jayjonah8, Jujic, kenzo, robee, sirhashalot, and WatchPug*
- [L-05] BorrowerOperations.withdrawColl doesn't check the length of the caller supplied arrays *Submitted by hyh*
- [L-06] User facing BorrowerOperations and TroveManager miss emergency lever *Submitted by hyh*
- [L-07] BorrowerOperations has unused pieces of functionality *Submitted by hyh, also found by heiho1*
- [L-08] sYETIToken rebase comment should be 'added is not more than repurchased' *Submitted by hyh*
- [L-09] WJLP setAddresses initialization can be front run *Submitted by hyh, also found by cmichel, Ruhum, and WatchPug*
- [L-10] BorrowerOperations and StabilityPool trove status check depends on the enumeration order *Submitted by hyh*
- [L-11] Target pool does not get updated due to receiveCollateral not being called *Submitted by csanuragjain*
- [L-12] Mixed compiler versions *Submitted by p4st13r4, also found by certora, robee, and WatchPug*
- [L-13] Incompatibility With Rebasing/Deflationary/Inflationary tokens *Submitted by defsec*
- [L-14] Ownable doesn't allow transferring ownership *Submitted by Ruhum*

- [L-15] contracts/TroveManagerLiquidations.sol is missing inheritance *Submitted by heiho1*
- [L-16] contracts/TroveManagerRedemptions.sol is missing inheritance *Submitted by heiho1*
- [L-17] Missing duplicate checks in `withdrawColl` *Submitted by cmichel, also found by gpersoon*
- [L-18] Missing cutoff checks in `adjustParams` *Submitted by cmichel*
- [L-19] No sanity check of safe ratio when adding collateral *Submitted by kenzo, also found by shenwilly*
- [L-20] Lack of precision *Submitted by certora*
- [L-21] CollSurplusPool doesn't verify that the passed `_whitelistAddress` is an actual contract addres *Submitted by Ruhum*
- [L-22] setAddresses should only be callable once *Submitted by pauliax*
- [L-23] Deleting a mapping within a struct *Submitted by pauliax*
- [L-24] Wrong vesting schedule for YETI mentioned in LockupContract *Submitted by kenzo*
- [L-25] `YetiFinanceTreasury.sol#updateTeamWallet()` should implement two-step transfer pattern *Submitted by WatchPug, also found by 0x1f8b, defsec, Ruhum, and shenwilly*
- [L-26] `ERC20_8.sol` `totalSupply` should be increased on `mint` and decreased on `burn` *Submitted by WatchPug, also found by kenzo and pauliax*
- [L-27] TeamLockup releases more tokens that it should *Submitted by kenzo*
- [L-28] Tokens with fee on transfer are not supported *Submitted by WatchPug*
- [L-29] Unsafe approve in sYETIToken *Submitted by 0x1f8b*
- [L-30] Attacker can steal future rewards of `WJLP` from other users *Submitted by WatchPug*
- [L-31] exists check passes when validCollateral length is 0 *Submitted by pauliax*
- [L-32] claimYeti inclusive check *Submitted by pauliax*
- [L-33] Missing return statements *Submitted by pauliax*

Non-Critical Findings (17)

- [N-01] WJLP.sol does not make use of important events to emit *Submitted by jayjonah8*
- [N-02] Use of Large Number Literals *Submitted by SolidityScan*
- [N-03] Missing events in critical functions *Submitted by SolidityScan*
- [N-04] Deprecated collateral check is missing in sendCollaterals *Submitted by csanuragjain*
- [N-05] StabilityPool does not update rewards when upwrapping wrapped asset *Submitted by kenzo*
- [N-06] Wrong assumption that wrapped asset holder is receiver of wrapped asset rewards *Submitted by kenzo*
- [N-07] NamespaceCollision: Multiple SafeMath contracts *Submitted by heiho1*
- [N-08] `lastFeeTime` can be reset *Submitted by cmichel*
- [N-09] `lastFeePercent` can be > 100% *Submitted by cmichel*
- [N-10] sYETIToken does not emit Approval event in `transferFrom` *Submitted by cmichel*
- [N-11] TODOs *Submitted by pauliax, also found by certora, Dravee, and robee*
- [N-12] Rescue assets in treasury contract *Submitted by pauliax*
- [N-13] ecrecover 0 address *Submitted by pauliax*
- [N-14] Race condition on ERC20 approval *Submitted by WatchPug, also found by cccz, certora, jayjonah8, and robee*
- [N-15] Missing error messages in require statements *Submitted by WatchPug, also found by certora and robee*
- [N-16] `_redeemCaller` should not obtain rights to future rewards for the `WJLP` they redeemed *Submitted by WatchPug*
- [N-17] Multiple contracts or interfaces with the same name *Submitted by heiho1*
- [N-18] Infinite mint *Submitted by 0x1f8b, also found by dalgarim* (Note: this issue was originally judged as high-risk, but later downgraded to non-critical.)

Gas Optimizations (56)

- [G-01] Unused imports *Submitted by robee, also found by WatchPug*
- [G-02] Short the following require messages *Submitted by robee*
- [G-03] Storage double reading. Could save SLOAD *Submitted by robee*
- [G-04] State variables that could be set immutable *Submitted by robee*
- [G-05] Unnecessary array boundaries check when loading an array element twice *Submitted by robee*
- [G-06] Prefix increments are cheaper than postfix increments *Submitted by robee, also found by 0x1f8b, certora, defsec, Dravee, Jujic, and WatchPug*
- [G-07] Unnecessary payable *Submitted by robee*
- [G-08] Named return issue *Submitted by robee, also found by certora and WatchPug*
- [G-09] Unused functions can be removed to save gas *Submitted by SolidityScan*
- [G-10] Long Revert Strings *Submitted by Jujic, also found by defsec, p4st13r4, pauliax, sirhashalot, and WatchPug*
- [G-11] Consider removing BaseBoringBatchable.sol *Submitted by jayjonah8*
- [G-12] Wrapped Joe LP token Contract JLP token variable is set on initialization, doesn't change afterwards and should be immutable *Submitted by Jujic*
- [G-13] Caching variables *Submitted by Jujic*
- [G-14] Remove GasPool.sol since its not needed *Submitted by jayjonah8*
- [G-15] Useless imports *Submitted by Jujic*
- [G-16] Avoid unnecessary storage read can save gas *Submitted by Jujic*
- [G-17] Upgrading the solc compiler to >=0.8 may save gas *Submitted by Jujic, also found by defsec, Dravee, and WatchPug*
- [G-18] Unnecessary use of Safemath *Submitted by Jujic*
- [G-19] A variable is being assigned its default value which is unnecessary. *Submitted by Jujic, also found by WatchPug*
- [G-20] Delete - ABI Coder V2 For Gas Optimization *Submitted by defsec*
- [G-21] Consider making some constants as non-public to save gas *Submitted by Jujic, also found by WatchPug*
- [G-22] uint is always >= 0 *Submitted by Jujic, also found by Dravee*

- [G-23] Checking zero address on msg.sender is impractical *Submitted by dalgarim, also found by cmichel and p4st13r4*
- [G-24] Debug code left over in WJLP.unwrapFor *Submitted by hyh, also found by p4st13r4 and WatchPug*
- [G-25] Less than 256 uints are not gas efficient *Submitted by defsec*
- [G-26] Use immutable *Submitted by 0x1f8b, also found by WatchPug*
- [G-27] Gas saving in ShortLockupContract *Submitted by 0x1f8b*
- [G-28] Gas savings *Submitted by csanuragjain*
- [G-29] Gas savings: Require statement is not needed *Submitted by csanuragjain*
- [G-30] Gas saving *Submitted by csanuragjain*
- [G-31] Usage of assert() instead of require() *Submitted by Dravee, also found by certora, certora, sirhashalot, and WatchPug*
- [G-32] Declare state variables as immutable *Submitted by p4st13r4*
- [G-33] Use `calldata` instead of `memory` for function parameters *Submitted by defsec, also found by Dravee*
- [G-34] Use of uint8 for counter in for loop increases gas costs *Submitted by Dravee*
- [G-35] Bytes constants are more efficient than string constants *Submitted by Dravee, also found by robee*
- [G-36] Explicit initialization with zero not required *Submitted by Dravee, also found by robee and sirhashalot*
- [G-37] Check if transfer amount > 0 *Submitted by Dravee, also found by WatchPug*
- [G-38] != 0 costs less gass compared to > 0 for unsigned integer inside pure or view functions *Submitted by Dravee, also found by defsec*
- [G-39] “constants” expressions are expressions, not constants, so constant `keccak` variables results in extra hashing (and so gas). *Submitted by Dravee, also found by pauliax*
- [G-40] contracts/Dependencies/CheckContract.sol has a potential gas optimization *Submitted by heiho1*

- [G-41] Gas: Unnecessary deadline increase *Submitted by cmichel, also found by certora, defsec, and WatchPug*
- [G-42] GAS: packing structs saves gas *Submitted by Ruhum, also found by 0x1f8b and robee*
- [G-43] `WJLP.getPendingRewards()` should be aview function *Submitted by Ruhum*
- [G-44] SafeMath with Solidity 0.8 *Submitted by pauliax, also found by jayjonah8, kenzo, and WatchPug*
- [G-45] Adding unchecked directive can save gas *Submitted by WatchPug*
- [G-46] Public functions not used by current contract should be external *Submitted by WatchPug, also found by cccz, heiho1, robee, and SolidityScan*
- [G-47] `10 ** 18` can be changed to `1e18` and save some gas *Submitted by WatchPug, also found by robee*
- [G-48] `HintHelpers.sol#setAddresses()` can be replaced with `constructor` and save gas *Submitted by WatchPug*
- [G-49] Inline unnecessary function can make the code simpler and save some gas *Submitted by WatchPug*
- [G-50] Only use `amount` when needed can save gas *Submitted by WatchPug*
- [G-51] Only using `SafeMath` when necessary can save gas *Submitted by WatchPug*
- [G-52] Gas Optimization: Unnecessary variables *Submitted by gzeon*
- [G-53] Cache array length in for loops can save gas *Submitted by WatchPug, also found by certora, defsec, Dravee, Jujic, and robee*
- [G-54] Cache storage variables in the stack can save gas *Submitted by WatchPug*
- [G-55] Cache repeated calculations *Submitted by pauliax*
- [G-56] _isBeforeFeeBootstrapPeriod inside the loop *Submitted by pauliax*

WJLP / Wrapped Assets

Because a significant number of findings in this contest relate to a wrapped asset subsystem (and in particular, the WJLP contract) that has subsequently been removed from the Yeti Finance protocol, we have listed these findings separately in

this report. At the time of the C4 audit contest launch, this subsystem was known to be not well tested and experimental, as noted in the contest repo; therefore the wardens were advised to consider it as an example only.

Of these WJLP / wrapped asset findings, 6 were high risk and 4 were medium risk.

High Risk WJLP/Wrapped Assets Findings (6)

[WH-01] _from and _to can be the same address on wrap() function

Submitted by jayjonah8

Impact

In WJLP.sol, the wrap() function pulls in _amount base tokens from _from, then stakes them to mint WAssets which it sends to _to. It then updates _rewardOwner's reward tracking such that it now has the right to future yields from the newly minted WAssets. But the function does not make sure that _from and _to are not the same address and failure to make this check in functions with transfer functionality has lead to severe bugs in other protocols since users rewards are updated on such transfers this can be used to manipulate the system.

Proof of Concept

https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/AssetWrappers/WJLP/WJLP.sol#L126

https://medium.com/@Knownsec_Blockchain_Lab/knownsec-blockchain-lab-i-kill-myself-monox-finance-security-incident-analysis-2dcb4d5ac8f

Recommended Mitigation Steps

require(address(_from) != address(_to), "_from and _to cannot be the same")

kingyetifinance (Yeti finance) confirmed:

> @LilYeti : Originally disputed since we had different wJLP on our main where this is already removed. part of this functionality is intended but the ability to frontrun the approve / wrap call is where the liability is.

[WH-02] WJLP will continue accruing rewards after user has unwrapped his tokens

Submitted by kenzo

WJLP doesn't update the inner accounting (for JOE rewards) when unwrapping user's tokens. The user will continue to receive rewards, on the expanse of users who haven't claimed their rewards yet.

Impact

Loss of yield for users.

Proof of Concept

The unwrap function just withdraws JLP from MasterChefJoe, burns the user's WJLP, and sends the JLP back to the user. It does not update the inner accounting (`userInfo`). **(Code ref)**

```
function unwrapFor(address _to, uint _amount) external overr
    _requireCallerIsAPorSP();
    _MasterChefJoe.withdraw(_poolPid, _amount);
    // msg.sender is either Active Pool or Stability Pool
    // each one has the ability to unwrap and burn WAssets t
    // send them to someone else
    _burn(msg.sender, _amount);
    JLP.transfer(_to, _amount);
}
```

Recommended Mitigation Steps

Need to keep userInfo updated. Have to take into consideration the fact that user can choose to set the reward claiming address to be a different account than the one that holds the WJLP.

kingyetifinance (Yeti finance) disagreed with severity and confirmed

> @LilYeti : Due to the specific issue mentioned here, it is actually different than #141 . If indeed the token is transferred before sending it to the protocol, it will be accepted as collateral but will never be able to leave the system, via liquidation,

> redemption, and should not have been able to leave except the flawed implementation of unwrap for in borrower operations allows this to be withdrawn. Large error nonetheless, recommend upgrading to severity 3.

[WH-03] WJLP loses unclaimed rewards when updating user's rewards

Submitted by kenzo, also found by UncleGrandpa925

After updating user's rewards in `_userUpdate`, if the user has not claimed them, and `_userUpdate` is called again (eg. on another `wrap`), the user's unclaimed rewards will lose the previous unclaimed due to wrong calculation.

Impact

Loss of yield for user.

Proof of Concept

When updating the user's unclaimedJoeReward, the function doesn't save it's previous value. **(Code ref)**

```
if (user.amount > 0) {
    user.unclaimedJOEReward = user.amount.mul(accJoePer
}
if (_isDeposit) {
    user.amount = user.amount.add(_amount);
} else {
    user.amount = user.amount.sub(_amount);
}
// update for JOE rewards that are already accounted for
user.rewardDebt = user.amount.mul(accJoePerShare).div(1e
```

So for example, rewards can be lost in the following scenario. We'll mark "acc1" for the value of "accJoePerShare" at step 1.

1. User Zebulun wraps 100 tokens. After `_userUpdate` is called: unclaimedJOEReward = 0, rewardDebt = 100*acc1.
2. Zebulun wraps 50 tokens: unclaimedJOEReward = 100*acc2* - *100*acc1, rewardDebt = 150 * acc2.

3. Zebulun wraps 1 token: unclaimedJOEReward = 150*acc3 - 150*acc2, rewardDebt = 151*acc3

So in the last step, Zebulun's rewards only take into account the change in accJoePerShare in steps 2-3, and lost the unclaimed rewards from steps 1-2.

Recommended Mitigation Steps

Change the unclaimed rewards calculation to:

```
user.unclaimedJOEReward = user.unclaimedJOEReward.add(user.amour
```

kingyetifinance (Yeti finance) disagreed with severity and confirmed:

> @LilYeti : Probably severity 3 due to loss of allocation of funds for other users wrapping LP tokens.

0xtruco (Yeti finance) disputed:

> Actually not a problem, the accrued reward is updated as time goes on and should be overridden. Follows same logic from Master chef v2 here: https://snowtrace.io/address/0xd6a4F121CA35509aF06A0Be99093d08462f53052#code

alcueca (Judge) commented:

> @0xtruco, are you sure of what you are saying? To me the issue seems to still exist. Could you elaborate?

0xtruco (Yeti finance) confirmed:

> @alcueca Yes you are correct this actually is an issue. I initially thought that we were harvesting rewards just like TJ when users wrapped tokens but turns out that line is not there. This is in the new version of our code but was not in the version submitted at the time of the contest. Thanks for looking into it more! Back to high risk.

[WH-04] Wrapped JLP can be stolen

Submitted by cmichel, also found by kenzo, pauliax, and WatchPug

The `WJLP.wrap` function accepts a `from` parameter and a `to` parameter. The tokens are transferred from the `from` account to the `to` account:

```
    function wrap(uint _amount, address _from, address _to, address
        // @audit can frontrun and steal => use from=victim, to=atta
        JLP.transferFrom(_from, address(this), _amount);
        JLP.approve(address(_MasterChefJoe), _amount);

        // stake LP tokens in Trader Joe's.
        // In process of depositing, all this contract's
        // accumulated JOE rewards are sent into this contract
        _MasterChefJoe.deposit(_poolPid, _amount);

        // update user reward tracking
        _userUpdate(_rewardOwner, _amount, true);
        _mint(_to, _amount);
    }
```

When a user wants to wrap their JLP tokens, they first need to approve the contracts with their token and in a second transaction call the `wrap` function. However, an attacker can frontrun the actual `wrap` function and call their own `wrap(from=victim, to=attacker)` which will make the victim pay with their approved tokens but the WJLP are minted to the attacker.

Impact

WJLP tokens can be stolen.

Recommended Mitigation Steps

Always transfer from `msg.sender` instead of using a caller-provided `from` parameter.

[WH-05] ActivePool unwraps but does not update user state in WJLP

Submitted by cmichel, also found by UncleGrandpa925

Calling `WJLP.unwrap` burns WJLP, withdraws the amount from the master chef and returns the same amount of JLP back to the `to` address. However, it does not update the internal accounting in `WJLP` with a `_userUpdate` call.

This needs to be done on the caller side according to the comment in the `WJLP.unwrap` function:

> "Prior to this being called, the user whose assets we are burning should have their rewards updated"

This happens when being called from the `StabilityPool` but not when being called from the `ActivePool.sendCollateralsUnwrap`:

```
function sendCollateralsUnwrap(address _to, address[] memory _tc
    _requireCallerIsBOorTroveMorTMLorSP();
    require(_tokens.length == _amounts.length);
    for (uint i = 0; i < _tokens.length; i++) {
        if (whitelist.isWrapped(_tokens[i])) {
            // @audit this burns the tokens for _to but does not
            IWAsset(_tokens[i]).unwrapFor(_to, _amounts[i]);
            if (_collectRewards) {
                IWAsset(_tokens[i]).claimRewardFor(_to);
            }
        } else {
            _sendCollateral(_to, _tokens[i], _amounts[i]); // re
        }
    }
    return true;
}
```

Impact

The `unwrapFor` call withdraws the tokens from the Masterchef and pays out the user, but their user balance is never decreased by the withdrawn amount. They can still use their previous balance to claim rewards through `WJLP.claimReward` which updated their unclaimed joe reward according to the old balance. Funds from the WJLP pool can be stolen.

Recommended Mitigation Steps

As the comment says, make sure the user is updated before each `unwrap` call. It might be easier and safer to have a second authorized `unwrapFor` function that accepts a `rewardOwner` parameter, the user that needs to be updated.

kingyetifinance (Yeti finance) confirmed:

> @LilYeti : This is indeed an issue which would cause loss of rewards from wrapper contract usage.

[WH-06] Liquidation can be escaped by depositing a WJLP with `_rewardOwner != _borrower`

Submitted by WatchPug

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/TroveManagerLiquidations.sol#L409-L409

```
    _updateWAssetsRewardOwner(collsToUpdate, _borrower, yetiFinanceT
```

In `_liquidateNormalMode()`, WAsset rewards for collToRedistribute will accrue to Yeti Finance Treasury, However, if a borrower wrap `WJLP` and set `_rewardOwner` to other address, `_updateWAssetsRewardOwner()` will fail due to failure of `IWAsset(token).updateReward()`.

https://github.com/code-423n4/2021-12-yetifinance/blob/5f5bf61209b722ba568623d8446111b1ea5cb61c/packages/contracts/contracts/AssetWrappers/WJLP/WJLP.sol#L126-L138

```
    function wrap(uint _amount, address _from, address _to, address
        JLP.transferFrom(_from, address(this), _amount);
        JLP.approve(address(_MasterChefJoe), _amount);

        // stake LP tokens in Trader Joe's.
        // In process of depositing, all this contract's
        // accumulated JOE rewards are sent into this contract
        _MasterChefJoe.deposit(_poolPid, _amount);
```

```
        // update user reward tracking
        _userUpdate(_rewardOwner, _amount, true);
        _mint(_to, _amount);
    }
```

PoC

1. Alice `wrap()` some `JLP` to `WJLP` and set `_rewardOwner` to another address;
2. Alice deposited `WJLP` as a collateral asset and borrowed the max amount of YUSD;
3. When the liquidator tries to call `batchLiquidateTroves()` when Alice defaulted, the transaction will fail.

Recommendation

Consider checking if the user have sufficient reward amount to the balance of collateral in `BorrowerOperations.sol#_transferCollateralsIntoActivePool()`.

Medium Risk WJLP/Wrapped Assets Findings (4)

[WM-01] WJLP contract doesn't check for JOE and JLP token transfers success

Submitted by hyh

Impact

Transactions will not be reverted on failed transfer call, setting system state as if it was successful. This will lead to wrong state accounting down the road with a wide spectrum of possible consequences.

Proof of Concept

_safeJoeTransfer do not check for JOE.transfer call success: https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/AssetWrappers/WJLP/WJLP.sol#L268

_safeJoeTransfer is called by _sendJoeReward, which is used in reward claiming.

JOE token use transfer from OpenZeppelin ERC20: https://github.com/traderjoe-xyz/joe-core/blob/main/contracts/JoeToken.sol#L9

Which does return success code: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol#L113

Trader Joe also uses checked transfer when dealing with JOE tokens: https://github.com/traderjoe-xyz/joe-core/blob/main/contracts/MasterChefJoeV3.sol#L102

Also, unwrapFor do not check for JLP.transfer call success: https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/AssetWrappers/WJLP/WJLP.sol#L166

Recommended Mitigation Steps

Add a require() check for the success of JOE transfer in _safeJoeTransfer function and create and use a similar function with the same check for JLP token transfers

kingyetifinance (Yeti finance) confirmed

[WM-02] Reward not transferred correctly

Submitted by csanuragjain, also found by hyh and jayjonah8

Impact

Monetary loss for user

Proof of Concept

1. Navigate to contract at https://github.com/code-423n4/2021-12-yetifinance/blob/main/packages/contracts/contracts/AssetWrappers/WJLP/WJLP.sol
2. Let us see _sendJoeReward function

```
function _sendJoeReward(address _rewardOwner, address _to) inter
        // harvests all JOE that the WJLP contract is owed
        _MasterChefJoe.withdraw(_poolPid, 0);

        // updates user.unclaimedJOEReward with latest data from
        _userUpdate(_rewardOwner, 0, true);

        uint joeToSend = userInfo[_rewardOwner].unclaimedJOERewa
        userInfo[_rewardOwner].unclaimedJOEReward = 0;
        _safeJoeTransfer(_to, joeToSend);
    }
```

3. Lets say user reward are calculated to be 100 so _safeJoeTransfer is called with joeToSend as 100. Also user remaining reward becomes 0
4. Let us see _safeJoeTransfer function

```
function _safeJoeTransfer(address _to, uint256 _amount) internal
        uint256 joeBal = JOE.balanceOf(address(this));
        if (_amount > joeBal) {
            JOE.transfer(_to, joeBal);
        } else {
            JOE.transfer(_to, _amount);
        }
    }
```

5. If the reward balance left in this contract is 90 then _safeJoeTransfer will pass if condition and contract will transfer 90 amount. Thus user incur a loss of 100-90=10 amount (his remaining reward are already set to 0)

Recommended Mitigation Steps

If the reward balance is lower than user balance then contract must transfer reward balance in contract and make remaining user reward balance as (user reward balance - contract reward balance)

kingyetifinance (Yeti finance) disagreed with severity:

> @LilYeti: In #61 it has description and explanation why should be severity 1.

alcueca (Judge) commented:

> From #61:
>
> @LilYeti : MasterChef is a decently well trusted contract and all JLP rewards are distributed there. Fundamentally the number should not be off by any, if any will be dust, and this exists to protect in the worst case so at least some users can get JOE out. However it is a backstop and extra safety measure. In #137 the reward being off by 10 would require an additional bug somewhere else, or a failure of MasterChef.

[WM-03] Unused WJLP can't be simply unwrapped

Submitted by kenzo

WJLP can only be unwrapped from the Active Pool or Stability Pool. A user who decided to wrap his JLP, but not use all of them in a trove, Wouldn't be able to just unwrap them.

Impact

Impaired functionality for users. Would have to incur fees for simple unwrapping.

Proof of Concept

The unwrap functionality is only available from `unwrapFor` function, and that function is only callable from AP or SP. (Code ref)

```
    function unwrapFor(address _to, uint _amount) external override
            _requireCallerIsAPorSP();
```

Recommended Mitigation Steps

Allow anybody to call the function. As it will burn the holder's WJLP, a user could only unwrap tokens that are not in use.

kingyetifinance (Yeti finance) confirmed

> Added unwrap function

[WM-04] ActivePool does not update rewards before unwrapping wrapped asset

Submitted by kenzo

When ActivePool sends collateral which is a wrapped asset, it first unwraps the asset, and only after that updates the rewards. This should be done in opposite order. As a comment in WJLP's `unwrapFor` rightfully mentions - "Prior to this being called, the user whose assets we are burning should have their rewards updated".

Impact

Lost yield for user.

Proof of Concept

In ActivePool's `sendCollateralsUnwrap` (which is used throughout the protocol), it firsts unwraps the asset, and only afterwards calls `claimRewardFor` which will update the rewards: (Code ref)

```
    IWAsset(_tokens[i]).unwrapFor(_to, _amounts[i]);
    if (_collectRewards) {
            IWAsset(_tokens[i]).claimRewardFor(_to);
    }
```

`claimRewardFor` will end up calling `_userUpdate` : (Code ref)

```
    function _userUpdate(address _user, uint256 _amount, bool _isDep
        uint256 accJoePerShare = _MasterChefJoe.poolInfo(_poolPid).a
        UserInfo storage user = userInfo[_user];
        if (user.amount > 0) {
            user.unclaimedJOEReward = user.amount.mul(accJoePerShare
        }
        if (_isDeposit) {
            user.amount = user.amount.add(_amount);
        } else {
            user.amount = user.amount.sub(_amount);
        }
        user.rewardDebt = user.amount.mul(accJoePerShare).div(1e12);
```

```
    }
```

Now, as ActivePool has already called `unwrapFor` and has burnt the user's tokens, and let's assume they all were used as collateral, it means user.amount=0*, and the user's unclaimedJOEReward won't get updated to reflect the rewards from the last user update. This is why, indeed as the comment in `unwrapFor` says, user's reward should be updated prior to that.

*Note: at the moment `unwrapFor` doesn't updates the user's user.amount, but as I detailed in another issue, that's a bug, as that means the user will continue accruing rewards even after his JLP were removed from the protocol.

Recommended Mitigation Steps

Change the order of operations in `sendCollateralsUnwrap` to first send the updated rewards and then unwrap the asset. You can also consider adding to the beginning of `unwrapFor` a call to `_userUpdate(_to, 0, true)` to make sure the rewards are updated before unwrapping. Note: as user can choose to have JOE rewards accrue to a different address than the address that uses WJLP as collateral, you'll have to make sure you update the current accounts. I'll detail this in another issue.

kingyetifinance (Yeti finance) confirmed:

> @LilYeti: Thanks for the thorough explanation.

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