Amun contest
Findings & Analysis Report

2023-03-07

Table of contents

Overview

Please note: Code4rena is an organization that puts learning at the forefront of everything we do. Our rules and processes continue to develop over time, and older reports may reflect previous iterations of these rules and processes. For a more current representation of Code4rena's severity standardization rules and comprehensive judging criteria, we recommend browsing the reports from C4's most recent contests.

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 Amun smart contract system written in Solidity. The code contest took place between December 13—December 19 2021.

Wardens

30 Wardens contributed reports to the Amun contest:

1. WatchPug (jtp and ming)
2. Czar102
3. certora
4. csanuragjain
5. cmichel
6. kenzo
7. robee
8. harleythedog
9. JMukesh
10. gpersoon
11. hyh
12. defsec
13. pauliax
14. pedroais
15. jayjonah8
16. pmerkleplant
17. p4st13r4 (0xb4bb4 and 0x69e8)
18. gzeon
19. GiveMeTestEther
20. Ruhum
21. Jujic
22. itsmeSTYJ
23. yeOlde
24. 0x1f8b
25. sirhashalot
26. Dravee

27. 0x0x0x
28. saian
29. shenwilly
30. hubble (ksk2345 and shri4net)

This contest was judged by 0xleastwood.

Final report assembled by burgertime and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 35 unique vulnerabilities and 129 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, 10 received a risk rating in the category of MEDIUM severity, and 23 received a risk rating in the category of LOW severity.

C4 analysis also identified 44 non-critical recommendations and 50 gas optimizations.

Scope

The code under review can be found within the C4 Amun contest repository, and is composed of 45 smart contracts written in the Solidity programming language and includes 585 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] Unused ERC20 tokens are not refunded, and can be stolen by attacker

Submitted by WatchPug

Under certain circumstances, e.g. `annualizedFee` being minted to `feeBeneficiary` between the time user sent the transaction and the transaction being packed into the block and causing amounts of underlying tokens for each basketToken to decrease. It's possible or even most certainly that there will be some leftover basket underlying tokens, as `BasketFacet.sol#joinPool()` will only transfer required amounts of basket tokens from Join contracts.

However, in the current implementation, only the leftover inputToken is returned.

As a result, the leftover underlying tokens won't be returned to the user, which constitutes users' fund loss.

`SingleTokenJoinV2.sol` L57-L78

```
    function joinTokenSingle(JoinTokenStructV2 calldata _joinTokenSt
        external
    {
        // ######## INIT TOKEN #########
        IERC20 inputToken = IERC20(_joinTokenStruct.inputToken);

        inputToken.safeTransferFrom(
```

```
            msg.sender,
            address(this),
            _joinTokenStruct.inputAmount
        );

        _joinTokenSingle(_joinTokenStruct);

        // ######## SEND TOKEN #########
        uint256 remainingIntermediateBalance = inputToken.balanceOf(
            address(this)
        );
        if (remainingIntermediateBalance > 0) {
            inputToken.safeTransfer(msg.sender, remainingIntermediat
        }
    }
```

BasketFacet.sol **L143-L168**

```
    function joinPool(uint256 _amount, uint16 _referral)
        external
        override
        noReentry
    {
        require(!this.getLock(), "POOL_LOCKED");
        chargeOutstandingAnnualizedFee();
        LibBasketStorage.BasketStorage storage bs =
            LibBasketStorage.basketStorage();
        uint256 totalSupply = LibERC20Storage.erc20Storage().totalSu
        require(
            totalSupply.add(_amount) <= this.getCap(),
            "MAX_POOL_CAP_REACHED"
        );

        uint256 feeAmount = _amount.mul(bs.entryFee).div(10**18);

        for (uint256 i; i < bs.tokens.length; i++) {
            IERC20 token = bs.tokens[i];
            uint256 tokenAmount =
                balance(address(token)).mul(_amount.add(feeAmount)).
                    totalSupply
                );
            require(tokenAmount != 0, "AMOUNT_TOO_SMALL");
            token.safeTransferFrom(msg.sender, address(this), tokenA
        }
```

```
...
```

Furthermore, the leftover tokens in the `SingleTokenJoinV2` contract can be stolen by calling `joinTokenSingle()` with fake `outputBasket` contract and `swap.exchange` contract.

Recommended Mitigation Steps

Consider:

1. Calling `IBasketFacet.calcTokensForAmount()` first and only swap for exactly the desired amounts of tokens (like `SingleTokenJoin.sol`);
2. Or, refund leftover tokens.

loki-sama (Amun) acknowledged

[H-02] It might not be possible to withdraw tokens from the basket

Submitted by Czar102, also found by csanuragjain

Impact

When enough basket token owners exit, it will be impossible to exit pool with the last `MIN_AMOUNT` tokens because of **this** check. This will result in locking some tokens forever.

Recommended Mitigation Steps

Consider resigning from this check or performing it only for the owner balance, who would need to have at least `MIN_AMOUNT` tokens.

loki-sama (Amun) disagreed with severity

0xleastwood (Judge) commented:

> Nice find! I think this is valid:)

Medium Risk Findings (10)

[M-01] Function `joinTokenSingle` in `SingleTokenJoin.sol` and `SingleTokenJoinV2.sol` can be made to fail

Submitted by pmerkleplant, also found by certora, hyh, p4st13r4, pauliax, robee, and WatchPug

Impact

There's a griefing attack vulnerability in the function `joinTokenSingle` in `SingleTokenJoin.sol` as well as `SingleTokenJoinV2.sol` which makes any user transaction fail with "FAILED*OUTPUT*AMOUNT".

Proof of Concept

The `JoinTokenStruct` argument for `joinTokenSingle` includes a field `outputAmount` to indicate the amount of tokens the user should receive after joining a basket (see line 135 and 130).

However, this amount is compared to the contract's balance of the token and reverts if the amount is unequal.

If an attacker sends some amount of a basket's token to the contract, every call to this function will fail as long as the output token equals the attacker's token send.

Recommended Mitigation Steps

Refactor the `require` statement to expect at least the `outputAmount` of tokens, i.e. `require(outputAmount >= _joinTokenStruct.outputAmount)`.

loki-sama (Amun) confirmed

[M-02] Unchecked return value from low-level call()

Submitted by JMukesh, also found by certora

Impact

The return value of the low-level call is not checked, so if the call fails, the Ether will be locked in the contract. If the low level is used to prevent blocking operations, consider logging failed calls.

Proof of Concept

https://github.com/code-423n4/2021-12-amun/blob/98f6e2ff91f5fcebc0489f5871183566feaec307/contracts/basket/contracts/singleJoinExit/EthSingleTokenJoinV2.sol#L26

Recommended Mitigation Steps

add condition to check return value

loki-sama (Amun) confirmed and disagreed with severity

0xleastwood (Judge) commented:

> Nice find! I think this could be marked as `medium` as it leaks value from the protocol but it doesn't result in assets being lost directly. It requires `_INTERMEDIATE_TOKEN` to point to a contract which fails upon wrapping the ETH amount.

> So considering that `_INTERMEDIATE_TOKEN` must be improperly set, I will mark this as `medium`.

[M-03] It is possible to "uninitialize" `ERC20Facet` contract

Submitted by Czar102

Impact

The initialization status is defined by the name and symbol. It is possible it set them back to an empty string, uninitializing the contract and letting the `initialize(..)` function be called again. This way, the owner may, for example, hide minting additional tokens. Or, after accidentally setting name and symbol to empty strings, anyone can take control over the contract and mint any number of tokens.

In general, it shouldn't be possible to initialize more than once.

Recommended Mitigation Steps

Consider adding empty string checks in `setName(...)` and `setSymbol(...)` functions.

loki-sama (Amun) confirmed

[M-04] Annualized fee APY dependence on the frequency of executing a function

Submitted by Czar102

Impact

The APY of the annualized fee is dependent on the frequency of the execution of the `BasketFacet::chargeOutstandingAnnualizedFee()`. If it is called more frequently, the compounding is more frequent and the APY is higher. For less used baskets, the APY might be lower, because the compounding will happen at lower rate.

Recommended Mitigation Steps

Consider calculating the fee as the compounding was continous or with a constant compounding period.

loki-sama (Amun) acknowledged

Oxleastwood (Judge) commented:

> Nice find!

[M-05] `totalSupply` may exceed `LibBasketStorage.basketStorage().maxCap`

Submitted by Czar102, also found by gpersoon, gzeon, kenzo, and WatchPug

Impact

Total supply of the token may exceed the `maxCap` introduced. This can happen when a user wants to join the pool. The **check in** `BasketFacet::joinPool(...)` includes only the base amount, without fee. Thus, if fee is on and someone will want to create as many tokens as possible, the `totalSupply + _amount` will be set to `maxCap`. The call will succeed, but new tokens were also minted as the fee for `bs.feeBeneficiary` if `bs.entryFee` and `bs.entryFeeBeneficiaryShare` are nonzero. Thus, the number of tokens may exceed `maxCap`.

Recommended Mitigation Steps

Consider calculating `feeAmount` and `feeBeneficiaryShare` before the `require(...)` statement and check `totalSupply.add(_amount).add(feeBanficiaryShare) <= this.getCap()`.

loki-sama (Amun) acknowledged

[M-06] block.timestamp or deadline

Submitted by gpersoon, also found by kenzo and robee

Impact

Some functions, like rebalance() in RebalanceManagerV3 use _deadline as a time limit for swapExactTokensForTokens() Other functions, like _joinTokenSingle() of SingleTokenJoinV2.sol and _exit() of SingleNativeTokenExitV2() use block.timestamp, although a deadline field is present in the struct.

Possibly the deadline fields should have been used.

Proof of Concept

`RebalanceManagerV3.sol` **L158-L203**

```
    function rebalance(UnderlyingTrade[] calldata _swapsV2, uint256
    ...
            for (uint256 i; i < _swapsV2.length; i++) {
      ...
                for (uint256 j; j < trade.swaps.length; j++) {
                    ..
```

```
                _swapUniswapV2(swap.exchange,input,0, swap.path,
```

RebalanceManagerV3.sol **L63-L104**

```
    function _swapUniswapV2(...) {
            basket.singleCall(
                exchange,
                abi.encodeWithSelector(  IUniswapV2Router02(exchange
                0
            );
```

SingleTokenJoinV2.sol **L80-L112**

```
    struct JoinTokenStructV2 {
          ...
            uint256 deadline;
          ...
        }
    function _joinTokenSingle(JoinTokenStructV2 calldata _joinTokenS
          ...
                for (uint256 j; j < trade.swaps.length; j++) {
                    IPangolinRouter(swap.exchange).swapExactTokensFo
                }
            }
```

SingleNativeTokenExitV2.sol **L59-L88**

```
     struct ExitTokenStructV2 {
            ...
            uint256 deadline;
           ...
        }
    function _exit(ExitTokenStructV2 calldata _exitTokenStruct) inte
          ...
            for (uint256 i; i < _exitTokenStruct.trades.length; i++)
               ...
                for (uint256 j; j < trade.swaps.length; j++) {
                    ...
                    IPangolinRouter(swap.exchange).swapExactTokensFo
```

```
            }
        }
```

Recommended Mitigation Steps

Check whether the deadline fields should have been used. If so replace block.timestamp with the appropriate deadline

loki-sama (Amun) confirmed

[M-07] ERC20 return values not checked

Submitted by cmichel, also found by defsec, JMukesh, p4st13r4, and WatchPug

The `ERC20.transfer()` and `ERC20.transferFrom()` functions return a boolean value indicating success. This parameter needs to be checked for success. Some tokens do **not** revert if the transfer failed but return `false` instead.

See:

- `SingleNativeTokenExitV2.exit`'s `outputToken.transfer(msg.sender, outputTokenBalance);`
- `PieFactoryContract.bakePie`'s `pie.transfer(msg.sender, _initialSupply);`

Impact

Tokens that don't actually perform the transfer and return `false` are still counted as a correct transfer and the tokens remain in the `SingleNativeTokenExitV2` contract and could potentially be stolen by someone else.

Recommended Mitigation Steps

We recommend using OpenZeppelin's `SafeERC20` versions with the `safeTransfer` and `safeTransferFrom` functions that handle the return value check as well as non-standard-compliant tokens.

Oxleastwood (Judge) commented:

> Nice find! I think this is valid considering the extent basket tokens are used. It is more than likely that non-standard tokens will be utilised.

[M-08] SingleNativeTokenExitV2 assumes first exchange holds the outputToken

Submitted by kenzo, also found by cmichel and hyh

SingleNativeTokenExitV2 allows the user to exit and execute trades via multiple exchanges. When finishing the trades and sending a single output token back to the user, the contract takes that token from the last swap in the first exchange's trades. There is nothing in the struct that signifies this will be the output token, and this also impairs the exit functionality.

Impact

Let's say a basket only holds token TOKE, and user would like to exit to DAI. But there's no exchange with good liquidity for TOKE -> DAI. So the user crafts a trade to exchange TOKE for WOKE in exchange A, and then exchange WOKE for DAI in exchange B, to finally receive back DAI. The contract will not let him do so, as the output token is taken to be the output token of the first exchange - WOKE in our example.

Proof of Concept

In `exit`, the output token is taken to be the last token exchanged in the first exchange: **(Code ref)**

```
    address[] calldata path = _exitTokenStruct
                .trades[0]
                .swaps[_exitTokenStruct.trades[0].swaps.length - 1]
                .path;
            IERC20 outputToken = IERC20(path[path.length - 1]); //th
```

This manifests the issue I detailed above.

Recommended Mitigation Steps

Have the outputToken be a parameter supplied in ExitTokenStructV2.

loki-sama (Amun) acknowledged

[M-09] Failed transfer with low level call could be overlooked

Submitted by harleythedog

Impact

The `CallFacet.sol` contract has the function `_call` :

```
    function  _call(
            address  _target,
            bytes  memory  _calldata,
            uint256  _value
    ) internal {
            require(address(this).balance >= _value, "ETH_BALANCE_TO
            (bool success, ) = _target.call{value: _value}(_calldata
            require(success, "CALL_FAILED");
            emit  Call(msg.sender, _target, _calldata, _value);
    }
```

This function is utilized in a lot of different places. According to the Solidity docs, “The low-level functions `call` , `delegatecall` and `staticcall` return `true` as their first return value if the account called is non-existent, as part of the design of the EVM. Account existence must be checked prior to calling if needed”.

As a result, it is possible that this call will not work but `_call` will not notice anything went wrong. It could be possible that a user is interacting with an exchange or token that has been deleted, but `_call` will not notice that something has gone wrong and as a result, ether can become stuck in the contract. For this reason, it would be better to also check for the contract’s existence prior to executing `_target.call` .

For reference, see a similar high severity reported in a Uniswap audit here (report # 9): https://github.com/Uniswap/v3-core/blob/main/audits/tob/audit.pdf

Proof of Concept

See `_call` here: `CallFacet.sol` L108.

Recommended Mitigation Steps

To ensure tokens don't get stuck in edge case where user is interacting with a deleted contract, make sure to check that contract actually exists before calling it.

loki-sama (Amun) confirmed

[M-10] fees calculations are not accurate

Submitted by certora

after that fee is calculated, it is minted to the feeBeneficiary. simply minting the exact amount results lower fee than it should be.

Impact

feeBeneficiary will get less fees than it should.

Proof of Concept

let's assume that the basket assets are worth 1M dollars, and totalSupply = 1M. the result of `calcOutStandingAnnualizedFee` is 100,00 so the feeBeneficiary should get 100,00 dollars. however, when minting 100,00 the totalSupply will increase to 1,100,000 so they will own 100000/1100000 * (1M dollars) = 90909.09 dollars instead of 100k

loki-sama (Amun) acknowledged:

> This is mitigated by the feeBeneficiary diluting his own shares if he gets fees on his fees.

0xleastwood (Judge) asked:

> I'm not exactly sure if I understand what the warden is stating here. Could you confirm @loki-sama ?

loki-sama (Amun) confirmed:

> Ok, I myself misunderstood. He is correct that we don't get the full value. When we take a fee of 10% like from his example. What we do is mint 10% of the basket

> to ourselves. That 10% after minting is not holding 10% of the underling.

Low Risk Findings (23)

- [L-01] Rebalance manager can steal tokens *Submitted by cmichel*
- [L-02] Underflow possible in rebalance() of RebalanceManagerV3 *Submitted by gpersoon, also found by certora*
- [L-03] Don't use transfer() *Submitted by GiveMeTestEther, also found by certora, cmichel, JMukesh, kenzo, p4st13r4, pauliax, robee, Ruhum, and WatchPug*
- [L-04] token.approve() doesn't check return value *Submitted by sirhashalot, also found by defsec, GiveMeTestEther, JMukesh, robee, Ruhum, and WatchPug*
- [L-05] Same facet can be added several times *Submitted by cmichel, also found by certora*
- [L-06] Approve 0 first *Submitted by pauliax, also found by GiveMeTestEther, itsmeSTYJ, and robee*
- [L-07] Use safe math for solidity version <8 *Submitted by robee*
- [L-08] Lost annualized fees due to early division *Submitted by kenzo, also found by certora, cmichel, Czar102, gzeon, itsmeSTYJ, robee, and WatchPug*
- [L-09] Basket might be unusable after initialization due to _initialSupply = 0 *Submitted by kenzo*
- [L-10] ETH that is accidentally sent to a receive() function cannot be withdrawn *Submitted by GiveMeTestEther, also found by certora, defsec, hyh, kenzo, and pedroais*
- [L-11] Lack of access modifier in Initialize() *Submitted by JMukesh*
- [L-12] Tokens with fee on transfer are not supported *Submitted by WatchPug, also found by defsec, gzeon, and hyh*
- [L-13] input amount might be wrong *Submitted by certora*
- [L-14] mint and burn of PolygonERC20Wrapper *Submitted by pauliax, also found by certora*
- [L-15] rounding error not in favor of the system *Submitted by certora*
- [L-16] minimum amount is too high *Submitted by certora, also found by hyh*

- [L-17] `initialize` functions can be frontrun *Submitted by cmichel, also found by hyh and robee*
- [L-18] Wrapped native token is assumed as the `INTERMEDIATE_TOKEN` *Submitted by cmichel, also found by defsec and Ruhum*
- [L-19] `_joinTokenSingle` can easily fail *Submitted by cmichel*
- [L-20] Different formulas to calculate tokenAmount *Submitted by gpersoon*
- [L-21] addToken() require check is not consistent with others similar to it *Submitted by jayjonah8*
- [L-22] Users can be frontrunned with higher fees *Submitted by pedroais*
- [L-23] Incorrect comment in `LibBasketStorage.sol` *Submitted by pmerkleplant, also found by hyh*

Non-Critical Findings (44)

- [N-01] Two-step change of a critical parameter *Submitted by certora*
- [N-02] Two-step change of a critical parameter *Submitted by certora*
- [N-03] Owner can add more tokens than `MAX_TOKENS` in `BasketFacet` *Submitted by Czar102*
- [N-04] Open TODOs *Submitted by Dravee, also found by defsec, pauliax, and robee*
- [N-05] Missing zero address validation on setRebalanceManager function *Submitted by Dravee*
- [N-06] Possible Re-entrancy *Submitted by defsec, also found by jayjonah8*
- [N-07] Use of floating pragma *Submitted by saian*
- [N-08] Solidity compiler versions mismatch *Submitted by robee*
- [N-09] Two Steps Verification before Transferring Ownership *Submitted by robee*
- [N-10] Not verified function inputs of public / external functions *Submitted by robee*
- [N-11] Incorrect revert reason in `CallFacet::addCaller(...)` *Submitted by Czar102*
- [N-12] No minimum fees is defined *Submitted by JMukesh*

- [N-13] Basketfacet#calcOutStandingAnnualizedFee() uses 365 days instead of 365.25 days *Submitted by GiveMeTestEther*
- [N-14] Due to lack of input validation , self transfer can happen *Submitted by JMukesh*
- [N-15] _swapsV3 after the _swapsV2 *Submitted by pauliax*
- [N-16] `RebalanceManager.sol#setRebalanceManager()` should implement two-step transfer pattern *Submitted by WatchPug*
- [N-17] missing zero check *Submitted by certora*
- [N-18] pointless amountOutMin *Submitted by certora*
- [N-19] no validation on max cap *Submitted by certora*
- [N-20] high centralization *Submitted by certora*
- [N-21] emit Transfer on withdraw *Submitted by pauliax*
- [N-22] Missing checks if pairs equal tokens *Submitted by defsec*
- [N-23] Missing validation of address argument could indefinitely lock RebalanceManager contracts *Submitted by defsec*
- [N-24] Named return issue *Submitted by robee*
- [N-25] wrong comment in line 228 of RebalanceManager.sol *Submitted by robee*
- [N-26] Add zero-address check in `setRebalanceManager()` *Submitted by shenwilly*
- [N-27] _maxApprove user input *Submitted by pauliax*
- [N-28] safeApprove of openZeppelin is deprecated *Submitted by robee*
- [N-29] Lacking zero address checks *Submitted by p4st13r4*
- [N-30] callNoValue() function does not guard against zero address *Submitted by jayjonah8*
- [N-31] Use of floating pragmas *Submitted by jayjonah8*
- [N-32] Setting allowance to uint256(-1) is bad practice *Submitted by itsmeSTYJ*
- [N-33] Missing zero address check in setDefaultController *Submitted by hubble*
- [N-34] Missing zero address check in setRebalanceManager *Submitted by hubble*

- [N-35] In my opinion in the future, hopefully there will be improvements in ui/ux, for the features to be divided into several parts, for example home, swap, leverage and also stake/farm are separated so that it is convenient to use, and will be more detailed because of the separation. This feature is very meaningful to make it easier for users *Submitted by 0x1f8b*
- [N-36] Division by `10 ** 18` *Submitted by Czar102*
- [N-37] Lack of zero-address checks *Submitted by WatchPug*
- [N-38] getTokenInPool() naming is confusing *Submitted by jayjonah8*
- [N-39] Unnecessary struct in SingleNativeTokenExitV2 *Submitted by kenzo*
- [N-40] Typo in event name *Submitted by p4st13r4*
- [N-41] Unlocked compiler version *Submitted by p4st13r4*
- [N-42] Unused state variables in MintableERC20 *Submitted by p4st13r4*
- [N-43] Validate diamond implementation is not empty *Submitted by pauliax*
- [N-44] 10**18 = HUNDRED_PERCENT *Submitted by pauliax*

Gas Optimizations (50)

- [G-01] Loops can be implemented more efficiently *Submitted by 0x0x0x, also found by defsec, Dravee, gzeon, kenzo, pmerkleplant, robee, and WatchPug*
- [G-02] ++i is more gas efficient than i++ in loops forwarding *Submitted by defsec, also found by 0x1f8b, Jujic, robee, and WatchPug*
- [G-03] Gas saving *Submitted by 0x1f8b*
- [G-04] Assigning local variables to unchanging storage variables *Submitted by Czar102, also found by robee*
- [G-05] Add `contractOwner` to `canCall` in `CallFacet` *Submitted by Czar102*
- [G-06] Use id to manage itarable addresses *Submitted by Czar102*
- [G-07] Call function internally instead of externally *Submitted by Czar102*
- [G-08] Redundant rewriting to memory *Submitted by Czar102*
- [G-09] Assigning keccak operations to constant variables results in extra gas costs *Submitted by Dravee, also found by gzeon, kenzo, and pauliax*
- [G-10] BasketFace#calcOutStandingAnnualizedFee no need for safe subtraction *Submitted by GiveMeTestEther*

- [G-11] Upgrade pragma to at least 0.8.4 *Submitted by defsec, also found by Jujic, WatchPug, and WatchPug*
- [G-12] Gas can be saved in the `calcTokensForAmount()` loop *Submitted by Jujic*
- [G-13] Consider making some constants as non-public to save gas *Submitted by Jujic*
- [G-14] Only pass lockBlock as parameter instead of LibBasketStorage.basketStorage() *Submitted by Jujic*
- [G-15] Change function visibility from public to external *Submitted by Jujic, also found by robee*
- [G-16] Unused imports *Submitted by robee, also found by Jujic*
- [G-17] Use `IWrappedNativeToken.deposit()` can save some gas *Submitted by WatchPug*
- [G-18] `ReentryProtection.sol` Switching between 1, 2 instead of increasing `lockCounter` is more gas efficient *Submitted by WatchPug*
- [G-19] `10 ** 18` can be changed to `1e18` and save some gas *Submitted by WatchPug*
- [G-20] State variables that could be set immutable *Submitted by robee, also found by WatchPug*
- [G-21] Avoid repeated arithmetic operations in for loop can save gas *Submitted by WatchPug, also found by pauliax*
- [G-22] Avoid unnecessary storage read can save gas *Submitted by WatchPug*
- [G-23] Use short circuiting can save gas *Submitted by WatchPug*
- [G-24] Reuse operation results can save gas *Submitted by WatchPug*
- [G-25] Remove unnecessary variables can make the code simpler and save some gas *Submitted by WatchPug*
- [G-26] `SingleTokenJoin#joinTokenSingle()` Change `inputAmount` to `maxInputAmount` can avoid dust INTERMEDIATE_TOKEN and save gas *Submitted by WatchPug*
- [G-27] Avoid unnecessary external calls can save gas *Submitted by WatchPug*
- [G-28] Internal call is more efficient than external call *Submitted by WatchPug*

- [G-29] Gas: Intermediate min-return check in `_joinTokenSingle` is unnecessary *Submitted by cmichel, also found by itsmeSTYJ*
- [G-30] Intermediate max-input-amount check in `_joinTokenSingle` is unnecessary *Submitted by cmichel*
- [G-31] Gas savings *Submitted by csanuragjain*
- [G-32] Use of _msgSender() *Submitted by defsec*
- [G-33] Delete - ABI Coder V2 For Gas Optimization *Submitted by defsec*
- [G-34] Revert String Size Optimization *Submitted by defsec, also found by robee*
- [G-35] Use a constant instead of block.timestamp for the deadline *Submitted by defsec*
- [G-36] Less than 256 uints are not gas efficient *Submitted by defsec*
- [G-37] uniSwapLikeRouter or swap.exchange *Submitted by pauliax, also found by kenzo*
- [G-38] Zero transfers *Submitted by defsec*
- [G-39] Pack structs tightly *Submitted by pauliax*
- [G-40] uint8 index *Submitted by robee*
- [G-41] PolygonERC20Wrapper can emit burn event instead of calling ERC20 functions *Submitted by kenzo*
- [G-42] PolygonERC20Wrapper does not need to be ERC20 *Submitted by kenzo*
- [G-43] Locking logic in Rebalancers can be changed to avoid SSTOREs. *Submitted by p4st13r4*
- [G-44] Not used variables *Submitted by pauliax*
- [G-45] Subtraction in `ERC20Facet::decreaseApproval` could be "unchecked" *Submitted by pmerkleplant*
- [G-46] Don't initialize variables with default value *Submitted by pmerkleplant*
- [G-47] Unnecessary array boundaries check when loading an array element twice *Submitted by robee*
- [G-48] Removing redundant code can save gas (RebalanceManager V2 V3, BaskerFacet) *Submitted by ye0lde*
- [G-49] Unnecessary explicit returns (BaskerFacet) *Submitted by ye0lde*

- [G-50] BasketFacet: no need for safe subtraction *Submitted by GiveMeTestEther*

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