Gro Protocol
Findings & Analysis Report

2021-08-30

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 of the Gro Protocol smart contract system written in Solidity. The code contest took place between June 30 — July 7, 2021.

Wardens

7 Wardens contributed reports to the Gro code contest:

- cmichel
- 0xRajeev
- gpersoon
- shw
- pauliax
- a_delamo
- GalloDaSballo

This contest was judged by ghoul.sol.

Final report assembled by moneylegobatman and ninek.

Summary

The C4 analysis yielded an aggregated total of 31 unique vulnerabilities. All of the issues presented here are linked back to their original finding

Of these vulnerabilities, 4 received a risk rating in the category of HIGH severity, 6 received a risk rating in the category of MEDIUM severity, and 21 received a risk rating in the category of LOW severity.

C4 analysis also identified 54 non-critical recommendations.

Scope

The code under review can be found within the C4 Gro Protocol code contest repository is comprised of 61 smart contracts written in the Solidity programming language.

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 (4)

[H-01] implicit underflows

Submitted by gpersoon, also found by cmichel

There are a few underflows that are converted via a typecast afterwards to the expected value. If solidity 0.8.x would be used, then the code would revert.

- `int256(a-b)` where a and b are uint: For example, if `a=1` and `b=2`, then the intermediate result would be `uint(-1) == 2**256-1`
- `int256(-x)` where x is a uint. For example, if `x=1`, then the intermediate result would be `uint(-1) == 2**256-1`

It's better not to have underflows by using the appropriate typecasts. This is especially relevant when moving to solidity 0.8.x.

From `Exposure.sol` L178:

```
function sortVaultsByDelta(..)
..
    for (uint256 i = 0; i < N_COINS; i++) {
        // Get difference between vault current assets and vault
        int256 delta = int256(unifiedAssets[i] - unifiedTotalAss
```

From `PnL.sol` L112:

```
function decreaseGTokenLastAmount(bool pwrd, uint256 dollarAmou
..
emit LogNewGtokenChange(pwrd, int256(-dollarAmount)); // underf
```

From `Buoy3Pool.sol` L87:

```
function safetyCheck() external view override returns (bool) {
    ...
        _ratio = abs(int256(_ratio - lastRatio[i])); // underflo
```

Recommend replacing `int256(a-b)` with `int256(a)-int256(b)`, and replacing `int256(-x)` with `-int256(x)`

kristian-gro (Gro) confirmed but disagreed with severity:

> Confirmed and We've mitigated this issue in our release version.

ghoul-sol (Judge) commented:

> Majority of overflow listed above seems low risk with one exception of `safetyCheck`. Underflow is a real risk here. `safetyCheck` is run every time a deposit is made. Ratios can change and the change does not need to be substantial for it to overflow. For that reason it's a high risk.

[H-02] `Buoy3Pool.safetyCheck` is not precise and has some assumptions

Submitted by cmichel, also found by shw

The `safetyCheck` function has several issues that impact how precise the checks are:

1. Only checks if the `a/b` and `a/c` ratios are within `BASIS_POINTS`. By transitivity, `b/c` is only within `2 * BASIS_POINTS` if `a/b` and `a/c` are in range. For a more precise check whether both USDC and USDT are within range, `b/c` must be checked as well.
2. If `a/b` is within range, this does not imply that `b/a` is within range.
3. > "inverted ratios, a/b bs b/a, while producing different results should both reflect the same change in any one of the two underlying assets, but in opposite directions"
4. Example: `lastRatio = 1.0 ratio: a = 1.0, b = 0.8` => `a/b = 1.25`, `b/a = 0.8` If `a/b` was used with a 20% range, it'd be out of range, but `b/a` is in range.
5. The NatSpec for the function states that it checks Curve and an external oracle, but no external oracle calls are checked, both `_ratio` and `lastRatio` are only from Curve. Only `_updateRatios` checks the oracle.

To address this issue, it is recommended to check if `b/c` is within `BASIS_POINTS`.

kristian-gro (Gro) confirmed but disagreed with severity:

> Makes strong assumption about the range of possible values - small differences between a and b will result in small differences between a/b and b/a - Extreme cases are handled by emergency. Agree on b/c check

kristian-gro (Gro) commented:

> medium severity - will only cause stop of deposits/withdrawals against curve, work around to put in emergency mode

kristian-gro (Gro) commented:

> Acknowledged, but the differences between variables are in basis points, we've simulated flash loan manipulations of curve and come to the conclusion that this approximation has a sufficiently small error margin to not cause issues. The B/C check (usdc/usdt) has been added in release version.

ghoul-sol (Judge) commented:

> A possibility of stopping deposits or withdrawals deserves high risk.

[H-03] Incorrect use of operator leads to arbitrary minting of GVT tokens

Submitted by 0xRajeev, also found by pauliax and gpersoon

The `distributeStrategyGainLoss()` function distributes any gains or losses generated from a harvest and is expected to be called only by valid protocol vault adaptors. It is an externally visible function and the access control is indirectly enforced on `msg.sender` by checking that `vaultIndexes[msg.sender]` is a valid index range 1-4. However, the operator used in the `require()` is `||` instead of `&&`, which allows an arbitrary `msg.sender`, i.e. attacker, to bypass the check.

Scenario: An arbitrary non-vault address calling this function will get an index of 0 because of default mapping value in `vaultIndexes[msg.sender]`, which will fail the `> 0` check, but pass the `<= N_COINS + 1` check (`N_COINS = 3`) because `0 <= 4` which will allow control to go past this check.

Furthermore, on L362, `index=0` will underflow the -1 decrement (due to lack of `SafeMath.sub` and use of < 0.8.0 solc) and the index will be set to `(uint256_MAX - 1)`. This will allow execution to proceed to the "else" part of conditional meant for curve LP vault. Therefore, this will allow any random address to call this function with arbitrary values of gain/loss and distribute arbitrary gain/loss appearing to come from Curve vault.

The attack control flow:

- -> `Controller.distributeStrategyGainLoss(ARBITRARY_HIGH_VALUE_OF_GAIN, 0)`
- -> `index = 0` passes check for the `index <= N_COINS + 1` part of predicate on L357 in `Controller.sol`
- -> `index = uint256_MAX` after L362
- -> `gainUsd = ibuoy.lpToUsd(ARBITRARY_HIGH_VALUE_OF_GAIN);` on L371 in `Controller.sol`
- -> `ipnl.distributeStrategyGainLoss(gainUsd, lossUsd, reward);` on L376 in `Controller.sol`
- -> `(gvtAssets, pwrdAssets, performanceBonus) = handleInvestGain(lastGA, lastPA, gain, reward);` on L254 in `PnL.sol`
- -> `performanceBonus = profit.mul(performanceFee).div(PERCENTAGE_DECIMAL_FACTOR);` on L186 of `PnL.sol`
- -> `gvt.mint(reward, gvt.factor(gvtAssets), performanceBonus);` on L256 in `PnL.sol`

Recommend changing `||` to `&&` in `require()` on L357 of `Controller.sol` to prevent arbitrary addresses from going past this check. Or, consider exercising explicit access control for the authorized vault adaptors.

kristian-gro (Gro) confirmed

> Confirmed and Fix has been implemented in release version.

[H-04] `sortVaultsByDelta` doesn't work as expected

Submitted by gpersoon, also found by shw

The function `sortVaultsByDelta` doesn't always work as expected.

Suppose all the delta's are positive, and delta1 >= delta2 >= delta3 > 0. Then `maxIndex = 0`. And `(delta < minDelta (==0) )` is never true, so `minIndex = 0`.

Then (assuming `bigFirst==true`):

```
vaultIndexes[0] = maxIndex = 0
vaultIndexes[2] = minIndex = 0
vaultIndexes[1] = N_COINS - maxIndex - minIndex = 3-0-0 = 3
```

This is clearly not what is wanted, all `vaultIndexes` should be different and should be in the range [0..2]. This is due to the fact that `maxDelta` and `minDelta` are initialized with the value 0. This all could results in withdrawing from the wrong vaults and reverts (because `vaultIndexes` [1] is out of range).

`Exposure.sol` L178:

```
function sortVaultsByDelta(bool bigFirst,uint256 unifiedTotalAss
        uint256 maxIndex;
        uint256 minIndex;
        int256 maxDelta;
        int256 minDelta;
        for (uint256 i = 0; i < N_COINS; i++) {
            // Get difference between vault current assets and v
            int256 delta = int256(
                unifiedAssets[i] - unifiedTotalAssets.mul(target
            );
            // Establish order
            if (delta > maxDelta) {
                maxDelta = delta;
                maxIndex = i;
            } else if (delta < minDelta) {
                minDelta = delta;
                minIndex = i;
```

```
            }
        }
        if (bigFirst) {
            vaultIndexes[0] = maxIndex;
            vaultIndexes[2] = minIndex;
        } else {
            vaultIndexes[0] = minIndex;
            vaultIndexes[2] = maxIndex;
        }
        vaultIndexes[1] = N_COINS - maxIndex - minIndex;
    }
```

Recommend the following

1. Initializing `maxDelta` and `minDelta`:

```
int256 maxDelta = -2**255; // or type(int256).min when using a newe
int256 minDelta  = 2**255; // or type(int256).max when using a newe
```

2. Check that `maxIndex` and `minIndex` are not the same
3. require (`maxIndex` != `minIndex`);

kristian-gro (Gro) confirmed:

> Confirmed and Fix has been implemented in release version.

Medium Risk Findings (6)

[M-01] Usage of deprecated ChainLink API in `Buoy3Pool`

*Submitted by cmichel, also found by 0xRajeev and a*delamo_

The Chainlink API (`latestAnswer`) used in the `Buoy3Pool` oracle wrappers is deprecated:

> This API is deprecated. Please see API Reference for the latest Price Feed API.
> [Chainlink Docs]

It seems like the old API can return stale data. Checks similar to that of the new API using `latestTimestamp` and `latestRoundare` are needed, as this could lead to stale prices according to the Chainlink documentation:

- under current notifications: "if answeredInRound < roundId could indicate stale data."
- under historical price data: "A timestamp with zero value means the round is not complete and should not be used."

Recommend adding checks similar to `latestTimestamp` and `latestRoundare`

```
(
    uint80 roundID,
    int256 price,
    ,
    uint256 timeStamp,
    uint80 answeredInRound
) = chainlink.latestRoundData();
require(
    timeStamp != 0,
    "ChainlinkOracle::getLatestAnswer: round is not complete"
);
require(
    answeredInRound >= roundID,
    "ChainlinkOracle::getLatestAnswer: stale data"
);
require(price != 0, "Chainlink Malfunction");
```

kristian-gro (Gro) confirmed

> Confirmed and Fix has been implemented in release version.

[M-02] Safe addresses can only be added but not removed

Submitted by 0xRajeev, also found by pauliax

The `addSafeAddress()` takes an address and adds it to a "safe list". This is used in `eoaOnly()` to give exemption to safe addresses that are trusted smart contracts, when all other smart contacts are prevented from protocol interaction. The stated purpose is to allow only such partner/trusted smart contract integrations (project rep

mentioned Argent wallet as the only one for now but that may change) an exemption from potential flash loan threats. But if there is a safe listed integration that needs to be later disabled, it cannot be done. The protocol will have to rely on other measures (outside the scope of this contest) to prevent flash loan manipulations which are specified as an area of critical concern.

Scenario: A trusted integration/partner address is added to the safe list. But that wallet/protocol/DApp is later manipulated (by the project, its users or an attacker) to somehow launch a flash loan attack on the protocol. However, its address cannot be removed from the safe list and the protocol cannot prevent flash loan manipulations from that source because of its exemption. Contract/project will have to be redeployed.

Recommend changing `addSafeAddress()` to `isSafeAddress()` with an additional bool parameter to allow both the enabling *AND* disabling of safe addresses.

kristian-gro (Gro) confirmed but disagreed with severity:

> low risk - Made specifically for one partner in beta period, and planned to be removed. We added the removal function for sanity.

> Confirmed and Fix has been implemented in release version.

ghoul-sol (Judge) commented:

> I'll keep medium risk because this could put the protocol into a one way street and not being able to remove safe addresses is quite dangerous. Medium risk.

[M-03] `BaseVaultAdaptor` assumes `sharePrice` is always in underlying decimals

Submitted by cmichel

The two `BaseVaultAdaptor.calculateShare` functions compute `share = amount.mul(uint256(10)**decimals).div(sharePrice)`

```
    uint256 sharePrice = _getVaultSharePrice();
    // amount is in "token" decimals, share should be in "vault" dec
```

```
share = amount.mul(uint256(10)**decimals).div(sharePrice);
```

This assumes that the `sharePrice` is always in *token* decimals and that *token* decimals is the same as *vault* decimals.

Both these assumptions happen to be correct for Yearn vaults, but that will not necessarily be the case for other protocols. As this functionality is in the `BaseVaultAdaptor`, and not in the specific `VaultAdaptorYearnV2_032`, consider generalizing the conversion.

Integrating a token where the token or price is reported in a different precision will lead to potential losses as more shares are computed.

Because the conversion seems highly protocol-specific, it is recommended that `calculateShare` should be an abstract function (like `_getVaultSharePrice`) that is implemented in the specific adaptors.

- kristian-gro (Gro) confirmed

> Confirmed and shares have been removed from release version.

[M-04] Flash loan risk mitigation is optional and not robust enough

Submitted by 0xRajeev

The `switchEoaOnly()` allows the owner to disable `preventSmartContracts` (the project's plan apparently is to do so after the beta-period) which will allow any smart contract to interact with the protocol and potentially exploit any underlying flash loan vulnerabilities which are specified as an area of critical concern.

The current mitigation is to optionally prevent contracts, except whitelisted partner ones, from interacting with the protocol to prevent any flash loan manipulations. A more robust approach would be to add logic preventing multiple txs to the protocol from the same address/ `tx.origin` within the same block when smart contracts are allowed. This will avoid any reliance on trust with integrating partners/protocols.

Recommend adding logic that prevents multiple txs to the protocol from the same address and within the same block.

kristian-gro (Gro) acknowledged but disagreed with severity:

> Low-severity: This is a temporary blocker to not let SCs interact with gro-protocol, planned to be removed after beta as it might potentially stop other integrations (as per issue 51)

> Acknowledged, this is just a temporary block, and is planned to be removed in future releases - other protection exists to protect the system from flash loan manipulations.

ghoul-sol (Judge) commented:

> It looks like a low risk issue since it's a future problem and not something that is an immediate issue, however, it's not clear how the protocol will protect itself against flash loans after this temporary blocker is off. One of the critical protocol's concerns are flash loans manipulations therefore I think medium risk is justified here.

[M-05] Use of deprecated Chainlink function `latestAnswer`

Submitted by shw

According to Chainlink's documentation (Deprecated API Reference, Migration Instructions, and API Reference), the `latestAnswer` function is deprecated. This function does not throw an error if no answer has been reached, but instead returns 0, causing an incorrect price to be fed to the `Buoy3Pool`. See `Buoy3Pool.sol` L207 and L214-L216.

Recommend using the `latestRoundData` function to get the price instead. Also recommend adding 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(answeredInRound >= roundID, "...");
require(timeStamp != 0, "...");
```

kristian-gro (Gro) disagreed with severity:

> disagree with severity (Low risk) Issue would cause deposits and withdrawals to stop, no funds lost

kristian-gro (Gro) confirmed:

> Confirmed and shares have been removed from release version.

ghoul-sol (Judge) commented:

> In my opinion halting the protocol deserves medium risk. While no funds are lost, from brand perspective it's a second worst thing. Keeping as medium risk.

[M-06] Early user can break minting

Submitted by cmichel

The protocol computes a `factor` when minting (and burning) tokens, which is the exchange rate of rebase to base tokens (base supply / total assets value), see `GToken.factor()`. The first user can manipulate this factor such that it always returns `0`.

Example:

- Attacker deposits 100.0 DAI and mints 100 * 1e18 PWRD: `DepositHandler.depositGToken` with `dollarAmount = 100.0 = 100 * 1e18`, then `ctrl.mintGToken(pwrd, msg.sender, 1e18)` calls `gt.mint(account, gt.factor(), amount=1e18)` where `gt.factor()` returns `getInitialBase() = 1e18` because the person is the first minter and it mints `amount * factor / _BASE = 1e18`
- The `ctrl.mintGToken` call also increases total assets: `pnl.increaseGTokenLastAmount(...)`
- The attacker now burns (withdraws) all minted tokens again **except a single wei** using one of the withdrawal functions in `WithdrawHandler`. Because of the withdrawal fee the total assets are only decreased by the post-fee amount (`IPnL(pnl).decreaseGTokenLastAmount(pwrd, amount=userBalance - 1,`

`bonus=fee);`), i.e., with a 2% withdrawal fee the total assets stay at 2% of 100$ = 2 * 1e18.

- The result is that `GToken.factor()` always returns `totalSupplyBase().mul(BASE).div(totalAssets) = 1 * 1e18 / (2 * 1e18) = 0`

The resulting `factor` is 0 and thus any user-deposits by `depositGToken` will mint 0 base tokens to the depositor. This means all deposits and future value accrues to the attacker who holds the only base tokens.

An attacker could even front-run the first minter to steal their deposit this way.

Uniswap solves a similar problem by sending the first 1000 tokens to the zero address which makes the attack 1000x more expensive. The same should work here, i.e., on first mint (`total base supply == 0`), lock some of the first minter's tokens by minting ~1% of the initial amount to the zero address instead of to the first minter.

kristian-gro (Gro) acknowledged but disagreed with severity:

> Known issue which will be handled by ops - low risk as gro protocol is the first depositor

ghoul-sol (Judge) commented:

> Even though it's a known issue its consequences are significant. Only because it can be mitigated by ops quite easily, I'll degrade it to medium level.

Low Risk Findings

[L-01] `emergencyHandler` not checked & not emitted

Submitted by gpersoon, also found by shw

The function `setWithdrawHandler` allows the setting of `withdrawHandler` and `emergencyHandler` . However, `emergencyHandler` isn't checked for 0 (like the `withdrawHandler`). The value of the `emergencyHandler` is also not emitted (like the `withdrawHandler`)

`Controller.sol` L105:

```
    function setWithdrawHandler(address _withdrawHandler, address _e
        require(_withdrawHandler != address(0), "setWithdrawHandler:
        withdrawHandler = _withdrawHandler;
        emergencyHandler = _emergencyHandler;
        emit LogNewWithdrawHandler(_withdrawHandler);
    }
```

Recommend adding something like:

```
        require(_emergencyHandler!= address(0), "setEmergencyHandler
        event LogNewEmergencyHandler(address tokens);
        emit LogNewEmergencyHandler(_emergencyHandler);
```

- kristian-gro (Gro) confirmed

[L-02] `lastRatio` of `Buoy3Pool` is not initialized

Submitted by gpersoon

The values of `lastRatio` in the contract `Buoy3Pool.sol` are not initialized (thus they have a value of 0). If `safetyCheck()` would be called before the first time `_updateRatios` is called, then `safetyCheck()` would give unexpected results.

`Buoy3Pool.sol` L25:

```
    contract `Buoy3Pool` is FixedStablecoins, Controllable, IBuoy, I
    ...
        mapping(uint256 => uint256) lastRatio;

     function safetyCheck() external view override returns (bool) {
            for (uint256 i = 1; i < N_COINS; i++) {
                uint256 _ratio = curvePool.get_dy(int128(0), int128(
                _ratio = abs(int256(_ratio - lastRatio[i]));
                if (_ratio.mul(PERCENTAGE_DECIMAL_FACTOR).div(CURVE_
                    return false;
                }
```

```
            }
            return true;
        }

    function _updateRatios(uint256 tolerance) private returns (bool
        ...
            for (uint256 i = 1; i < N_COINS; i++) {
                lastRatio[i] = newRatios[i];
```

Recommend double checking if this situation can occur and perhaps calling `_updateRatios` as soon as possible. Or alternatively, check in `safetyCheck` that the `lastRatio` values are initialized.

- kristian-gro (Gro) acknowledged

[L-03] `Buoy3Pool._updateRatios` unsafe math

Submitted by cmichel

The function performs type conversions and subtraction without over-/underflow checks:

```
    uint256 check = abs(int256(_ratio) - int256(chainRatios[i].div(C
```

Recommend checking if the values fit within the type range first, otherwise revert with a meaningful error message, as well as checking for underflows.

ghoul-sol (Judge) commented:

> This is partially a duplicate of #6 but it focuses on low risk issue so I'll record is as a separate (low risk) issue.

[L-04] `setUnderlyingTokenPercent` should check percentages

Submitted by gpersoon, also found by cmichel

The function `setUnderlyingTokenPercent` doesn't check that the sum of all the percentages is 100%. This way the percentages could be accidentally set up the wrong way, with unpredictable results. Note that the function can only be called by controller or the owner so the likelihood of mistakes is pretty low.

`Insurance.sol` **#L100**:

```
    function setUnderlyingTokenPercent(uint256 coinIndex, uint256 pe
        require(msg.sender == controller || msg.sender == owner(), "
        underlyingTokensPercents[coinIndex] = percent;
        emit LogNewTargetAllocation(coinIndex, percent);
    }
```

Recommend changing `setUnderlyingTokenPercent` to set the percentages for all the coins at the same time. And check that the sum of the percentages is 100%

[L-05] initialize `maxPercentForWithdraw` and `maxPercentForDeposit` ?

Submitted by gpersoon

The parameters `maxPercentForWithdraw` and `maxPercentForDeposit`, which are not directly initialized, will work in a suboptimal way If, the functions which rely on these parameters, are called before `setWhaleThresholdWithdraw` / `setWhaleThresholdDeposit`.

`Insurance.sol` **L63**:

```
    uint256 public maxPercentForWithdraw;
    uint256 public maxPercentForDeposit;

    function setWhaleThresholdWithdraw(uint256 _maxPercentForWithdra
        maxPercentForWithdraw = _maxPercentForWithdraw;
        emit LogNewVaultMax(false, _maxPercentForWithdraw);
    }
    function setWhaleThresholdDeposit(uint256 _maxPercentForDeposit)
        maxPercentForDeposit = _maxPercentForDeposit;
        emit LogNewVaultMax(true, _maxPercentForDeposit);
```

```
    }
```

Recommend assigning a default value to `maxPercentForWithdraw` and `maxPercentForDeposit`. Or alternatively, initializing the values via the constructor.

kristian-gro (Gro) acknowledged:

> This is known and values are updated as part of deployment scripts

[L-06] use `safemath`

Submitted by gpersoon

`Safemath` is used in several places but not everywhere. Especially in risky places like `PnL` and `distributeStrategyGainLoss` where it is hardly worth the gas-savings of not using `safemath`.

In `distributeStrategyGainLoss` it does make a difference, also due to another issue.

`PnL.sol` **L215**:

```
    function handleLoss( uint256 gvtAssets, uint256 pwrdAssets, uint
            uint256 maxGvtLoss = gvtAssets.sub(DEFAULT_DECIMALS_FACT
            if (loss > maxGvtLoss) {
              ...
            } else {
                gvtAssets = gvtAssets - loss;     // won't underflow
            }

        function forceDistribute() private {
            uint256 total = _controller().totalAssets();
            if (total > lastPwrdAssets.add(DEFAULT_DECIMALS_FACTOR))
                lastGvtAssets = total - lastPwrdAssets;    // won't u
            } else {
       ...
```

`Controller.sol` **L355**

```
function distributeStrategyGainLoss(uint256 gain, uint256 loss) {
    uint256 index = vaultIndexes[msg.sender];
    require(index > 0 || index <= N_COINS + 1, "!VaultAdaptor");
    ..
    index = index - 1;  // can underflow
```

Recommend applying `safemath` or moving to Solidity 0.8.x

- kristian-gro (Gro) acknowledged

[L-07] Missing emits for declared events

Submitted by 0xRajeev, also found by cmichel

Missing emits for declared events indicate potentially missing logic, redundant declarations, or reduced off-chain monitoring capabilities.

Scenario: For example, the event `LogFlashSwitchUpdated` is missing an emit in `Controller.sol`. Based on the name, this is presumably related to flash loans being enabled/disabled which could have significant security implications. Or the (misspelled) `LogHealhCheckUpdate` which is presumably related to a health check logic that is missing in `LifeGuard`. See issue page for referenced code.

Recommend evaluating if logic is missing and if so, adding logic+emit or removing event.

kristian-gro (Gro) confirmed

[L-08] Having only owner unpause/restart is risky

Submitted by 0xRajeev

The design choice seems to allow a whitelist of addresses (bots or trusted parties) that can trigger pause/emergency but `onlyOwner` can unpause/restart (and perform other privileged functions). While it is recommended in general to have separate privileges/roles for stopping and starting critical functions, having only a single owner for unpause/restart triggering may create a single point of failure if owner is EOA and keys are lost/compromised.

Scenario: `Protocol` is paused or put in emergency mode by a bot/user in whitelist. `Owner` is an EOA and the private keys are lost. `Protocol` cannot be unpaused/restarted. See `Controller.sol` #L317, #L101, and #L97.

Recommend evaluating this design choice to see if a whitelist should also be allowed to unpause/restart. At a minimum, use a 6-of-9 or higher multisig and not an EOA.

kristian-gro (Gro) acknowledged:

> Multi sig planned

[L-09] Uninitialized vaults/addresses will lead to reverts

Submitted by OxRajeev

Uninitialized system/curve vaults (default to zero address) will lead to reverts on calls and expect owner to set them before other functions are called because functions do not check if system has been initialized. This requires a robust deployment script which is fail-safe.

The same applies to many other address parameters in the protocol e.g.: `reward`.

Scenario: All vaults are not initialized because of a script error or an admin mistake. Protocol goes live and user interactions result in exceptions. Users lose trust and protocol reputation takes a hit. See `Controller.sol` #L136-L150 and #L194-L198.

Recommend evaluating non-zero defaults, initializing from constructor or maintaining/checking an initialization state variable which prevents other functions from being called until all critical system states such as vault addresses are initialized.

kristian-gro (Gro) acknowledged:

> Controller by gro governance, dealt with in deployment scripts

[L-10] The use of `tx.origin` for smart contract safe list is risky and not generic

Submitted by OxRajeev

The `addSafeAddress()` takes an address and adds it to a "safe list". This is used in `eoaOnly()` to give exemption to safe addresses that are trusted smart contracts, when all other smart contacts are prevented from protocol interaction. The stated purpose is to allow only such partner/trusted smart contract integrations (project rep mentioned Argent wallet as the only one for now but that may change) an exemption from potential flash loan threats.

The `eoaOnly()` check used during deposits and withdrawals checks if `preventSmartContracts` is enabled and if so, makes sure the transaction is coming from an integration/partner smart contract. But instead of using `msg.sender` in the check it uses `tx.origin`. This is suspect because `tx.origin` gives the originating EOA address of the transaction and not a smart contract's address. (This may get even more complicated with the proposed EIP-3074.)

Discussion with the project team indicated that this is indeed not the norm but is apparently the case for their only current (none others planned) integration with Argent wallet where the originating account is Argent's relayer `tx.origin` i.e. flow:

> Argent relayer (`tx.origin`) => Argent user wallet (`msg.sender`) => gro protocol while the typically expected flow is: user EOA (`tx.origin`) => proxy (`msg.sender`) => gro protocol.

While this has reportedly been verified and tested, it does seem strange and perhaps warrants a re-evaluation because the exemption for this/other trusted integration/partner smart contracts will not work otherwise.

Scenario: Partner contract is added to the safe address for exemption but the integration fails because of the use of `tx.origin` instead of `msg.sender`. See `Controller.sol` **#L266-L272**, **#L176-L178**, and **#L171-L174**.

Recommend re-evaluating the use of `tx.origin` instead of `msg.sender`.

kristian-gro (Gro) acknowledged:

> Made for specific partner, so cannot be generic

> *While this has reportedly been verified and tested, it does seem strange and perhaps warrants a re-evaluation because the exemption for this/other trusted*

> *integration/partner smart contracts will not work otherwise.*

> This SC protection is only temporary as we are aware of EIP-3074, and consideration of how we need to change this/or if we need this at all are ongoing.

[L-11] Missing parameter validation

Submitted by cmichel, also found by 0xRajeev

Some parameters of functions are not checked for invalid values:

- `BaseVaultAdaptor.constructor` : The addresses should be checked for non-zero values
- `LifeGuard3Pool.constructor` : The addresses should be checked for non-zero values
- `Buoy3Pool.constructor` : The addresses should be checked for non-zero values
- `PnL.constructor` : The addresses should be checked for non-zero values
- `Controllable.setController` : Does not check that `newController != controller`

A wrong user input, or wallets defaulting to the zero addresses for a missing input, can lead to the contract needing to redeploy or wasted gas.

Recommend validating the parameters.

kristian-gro (Gro) acknowledged:

> Low risk/Non critical - Deployment script handles these cases, but good practice to have 0x checks to stop wasting gas and having to redeploy.

[L-12] Stricter than needed inequalities may affect borderline scenarios

Submitted by 0xRajeev

Token amounts/prices are typically open-ranged and inclusive of the bounds. Using '<' or '>' instead of '<=' and '>=' may affect borderline scenarios, be considered

unintuitive by users, and affect accounting.

-**Scenario 1:** In `calculateVaultSwapData()`, the `require()` check is:

```
require(withdrawAmount < state.totalCurrentAssetsUsd, "Withdrawa
```

The '<' could be replaced by '<='

Scenario 2: In `withdrawSingleByLiquidity()`, the `require()` check is:

```
require(balance > minAmount, "withdrawSingle: !minAmount");
```

The '>' should be '>=' as is used in the similar check in `withdrawSingleByExchange()`.

See `Insurance.sol` L429, and `LifeGuard3Pool.sol` L224 and L268.

Recommend reconsidering strict inequalities and relaxing them if possible.

kristian-gro (Gro) acknowledged:

> We haven't been able to model an exploit for this

[L-13] `totalAssets > withdrawelUsd` should be inclusive

Submitted by pauliax

The check should be inclusive here to cover the case when `totalAssets = withdrawalUsd`:

Recommend changing

```
require(totalAssets > withdrawUsd, "totalAssets < withdrawalUsd"
```

to

```
require(totalAssets >= withdrawUsd, "totalAssets < withdrawalUsd
```

kristian-gro (Gro) acknowledged:

> Edge case that is unlikely to cause issues as gro protocol provides initial seed investment

[L-14] Use of uninitialized value and unclear/unused logic

Submitted by 0xRajeev, also found by cmichel

`vaultIndexes` is uninitialized and it's unclear what 10000 signifies here. `investDelta` return value is also ignored at call site. If this is an indication of missed/incorrect logic, then it's risky. If not, removing will help readability/maintainability. See `Insurance.sol` #L166, and #L155.

Recommend evaluating any missing logic or else removing unused code.

- kristian-gro (Gro) confirmed

[L-15] decimals of `FixedStablecoins`

Submitted by pauliax

The `FixedStablecoins` constructor does not validate that addresses in the array are not empty, `!= address(0)` , and instead relies on the creator passing the correct values for decimals. The comment next to USDC (0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48) says that it is supposed to have 6 decimals:

```
`uint256 public immutable USDC_DECIMALS; // = 1E6;`
```

However, when querying the actual value on Etherscan, it shows 0 decimals: The problem with USDC is that it uses a proxy pattern, thus the implementation could change (decimals could change but in practice, I think it is very unlikely).

I think it would be better not to pass decimals separately and, instead of relying on the correctness of the input, recommend using `IERC20Detailed` to query the decimals in code. Always querying the decimals on the go may be very inefficient and bring new attack vectors so I think you need to do here an assumption that decimals of upgradeable tokens won't change.

kristian-gro (Gro) acknowledged:

> We dont expect to see any changes to underlying decimals of stablecoins, worst case scenario we can redeploy the affected contracts

[L-16] More accurate calculation of return USD of `withdrawSingleByLiquidity`

Submitted by shw

The `withdrawSingleByLiquidity` function of `LifeGuard3Pool` calls `buoy.singleStableToUsd` to calculate the return USD amount, which internally calls `_stableToUsd` with the `deposit` parameter set to `true`. A more accurate calculation is to set the `deposit` parameter to `false` since this action is a withdrawal. A similar issue exists in the function `calcProtocolWithdraw` of `Allocation`, where the current strategy's USD is calculated by `buoy.singleStableToUsd`. See `LifeGuard3Pool.sol` **#L226**, `Buoy3Pool.sol` **#L122**, and `Allocation.sol` **#L142**.

Recommend considering adding a new boolean parameter, `deposit`, to the `singleStableToUsd` function of `Buoy3Pool` to indicate whether the action is a deposit or not, as that in the `stableToUsd` and `stableToLp` functions.

- **kristian-gro (Gro) confirmed**

[L-17] Use of `tx.origin` for authentication

Submitted by shw

The `eoaOnly` function of `Controller` checks whether the user is whitelisted using `tx.origin`. Using `tx.origin` to authenticate users is generally not a good practice

since it can be abused by malicious contracts when whitelisted users are interacting with them. Users have to be very careful to avoid being impersonated when interacting with contracts from other protocols, which could unnecessarily burden users. See `Controller.sol` **#L269**. For for more discussion on `tx.origin`, refer to **Solidity issue - Remove tx.origin**.

Recommend changing `tx.origin` at line 269 to `msg.sender` to ensure that the entity calling the `Controller` is the one allowed.

ghoul-sol (Judge) commented:

> This issue touches on different problem so I'll keep it as stand-alone low risk issue.

[L-18] Vault assets can be migrated to an arbitrary address at anytime by owner

Submitted by 0xRajeev, also found by gpersoon

`BaseVaultAdaptor` contains logic that is "built on top of any vault in order for it to function with Gro protocol." One of such functions is the `migrate()` function which is `onlyOwner` and takes an address parameter which allows owner to migrate the vault's entire balance at any time to that address. This is extremely risky because it gives an opportunity for, or at least a perception of, a rug-pull by a disgruntled/malicious owner/dev to the protocol users/community. This could also be dangerous if triggered accidentally, especially by an EOA owner address or maliciously via compromised keys.

- **Scenario 1:** Protocol launches and starts accumulating TVL. A savvy user analyzes source and shares the presence of this `migrate()` function as potential owner rug-pull vector. Users withdraw funds and protocol reputation takes a hit.
- **Scenario 2:** Protocol launches and hits 100MM TVL. A disgruntled dev/owner migrates vault assets to their address and drains the protocol.
- **Scenario 3:** Protocol launches and hits 100MM TVL. Owner EOA keys get compromised and attacker migrates vault assets to their address and drains the protocol.

See `BaseVaultAdaptor.sol` **#L294-L302**

See similar concern on `migrate()` functionality in ShibaSwap recently from Yearn devs here and here. Also from here and here.

Recommend evaluating the need for this function and then avoiding/mitigating the risk appropriately.

kristian-gro (Gro) confirmed but disagreed with severity:

> Low risk
>
> - Owner is timelock, plan for multi sig. -assumes malicious owner

ghoul-sol (Judge) commented:

> Agree with sponsor. Assuming malicious behavior of owner is low risk if it's a governance and timelock is used. Low risk.

[L-19] Enabling `preventSmartContracts` may lead to lock/loss of funds

Submitted by 0xRajeev

`preventSmartContracts` is initialized to false, which allows users to deposit/withdraw funds from the protocol via (custom) smart contracts because the `eoaOnly` check during deposits/withdrawals always succeeds. However, if protocol owner decides to suddenly enable `preventSmartContracts`, then smart contracts are prevented from interaction unless they are exempted in safe addresses.

The lack of an event in `switchEoaOnly()` to inform off-chain monitors `/interfaces` about the enabling/disabling, say from false -> true, and lack of a time-delayed enforcement of this prevention of contracts from depositing/withdrawing, causes users who have previously deposited via smart contracts (that are not `safeAddresses`) to get locked out of withdrawals leading to fund lock/loss.

Scenario: User deposits funds via smart contract (not in safe address list) when `preventSmartContracts` =false. Protocol owner sets `preventSmartContracts` =true. User's funds are locked/lost in protocol. See issue page for affected code.

Recommend adding event + time-delayed enforcement to `switchEoaOnly()` so users can monitor and withdraw funds deposited via smart contracts.

kristian-gro (Gro) disputed:

> Low criticality/Not an issue - Workaround exists (safe addresses)
>
> - Owner will be a timelock

ghoul-sol (Judge) commented:

> Agree with sponsor. While the scenario is correct, it all comes down to the management of the protocol. From different context I also assume that this option will be set to true for beta and safe addresses will be whitelisted. I'm making this a low risk because this can create too many angry users to be non-critical.

[L-20] Unauthorized `rebalanceTrigger` calls may allow one to exploit arbitrage opportunity and put system at risk

Submitted by 0xRajeev

The need for an externally visible `rebalanceTrigger()` (when `rebalance()` does that check itself) is apparently that the whitelisted bot checks trigger before calling the very expensive/security-sensitive `rebalance()` operation which again checks to see if anything has changed between then and the previous trigger.

Exposing the rebalance trigger check externally for convenience may offer a front-running arbitrage opportunity to a non-whitelisted, i.e. any, bot which can check when a rebalance will be triggered by a whitelisted bot and then using that information to arbitrage on underlying stablecoins/strategies, which may affect system exposure.

Discussion with the project team reported that this is technically possible, but only within the BP limit (25-50) of the current vs cached price (where the `BASIS_POINTS` is currently set to 20). If not, the `Buoy safetyCheck` will fail. See `Insurance.sol` #L187-L196 and #L198-L215. Also `Buoy3Pool.sol` #L30.

Recommend adding `onlyWhitelist` modifier to `rebalanceTrigger()`, which allows retaining the convenience of (only whitelisted) bots checking before calling

rebalance. This makes it only a little safer because one can always front-run the actual rebalance call. This will only force bots to monitor `mempool` for rebalances instead of arbing ahead of time. Revisit this aspect for any missed considerations.

kristian-gro (Gro) acknowleged and disagreed:

> There is price check before rebalance. It is not very useful to add whitelist on view function. Because the code is public, anyone can implement a local function easily.

ghoul-sol (Judge) commented:

> Solution proposed by warden does not solve the problem but the problem still remains valid. The issue looks quite generic and it's really a MEV problem that most protocols have. For that reason, I think it's reasonable to degrade to low risk.

[L-21] Rational actors will just set themselves as referral

Submitted by cmichel

When depositing, a referral can be chosen and the only check is:

```
account != address(0) && referral != address(0) && referrals[acc
```

This means that users can refer themselves. It's not immediately clear from the contracts that are part of this repo, what the referrals are used for. If they are used for anything, rational actors will always refer themselves to maximize profits making the referral system useless.

Recommend whitelisting big influencers that are allowed to be used as referrals to avoid everyone referring themselves or another account they control.

kristian-gro (Gro) disputed and disagreed with severity:

> not an issue/non-critical Makes no difference, referrals are calculated offchain and not used for anything on chain

ghoul-sol (Judge) commented:

> Even if this is calculated off-chain, technically being able to refer ourselves is an issue. Even offchain this needs to be filtered out which is extra work. I'm keeping this as low risk.

Non-Critical Findings (31)

- [N-01] hardcoded values
- [N-02] implicit assumptions about underlying coins
- [N-03] `setFeeToken` doesn't check index
- [N-04] redundant check of array length
- [N-05] Outdated comment at `calculateWithdrawalAmountsOnPartVaults`
- [N-06] require comments don't all follow convention
- [N-07] Easier way to determine `strategiesLength`
- [N-08] BASIS_POINTS naming convention
- [N-09] Unused code
- [N-10] Incorrect error strings used may cause confusion
- [N-11] `updateStrategiesDebtRatio` function and `LogNewDebtRatios` event
- [N-12] `setBigFishThreshold` above 100%
- [N-13] Inconsistent usage of exponentiation for constants
- [N-14] event `LogTransfer` is only emitted in function transfer
- [N-15] Missing input validation on `_feeToken` in `DepositHandler` constructor and `setFeeToken()`
- [N-16] Emergency disabling can only be done one stablecoin at a time
- [N-17] Whitelist addition/removal is done unconditionally
- [N-18] withdrawal fee may be set above 100% or frontrunned
- [N-19] `burnAll` should check that factor > 0 and amount > 0
- [N-20] Loss of precision
- [N-21] Hardcoded 99 as deadcoin
- [N-22] Wrong min amount check in `withdrawByStablecoin`
- [N-23] `Exposure.sortVaultsByDelta` does not work for N_COINS != 3

Gas Optimizations (24)

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