Brahma Console: Final Audit Report

- This is the client's copy for the final audit report for Brahma Console-Core and Brahma Console-Integrations.

Foreword

A security review of the *console-core* and *console-integrations* smart contracts from Brahma Finance was done by Rahul Saxena and Parth Patel.

This audit report includes all the vulnerabilities, issues, recommendations, gas and code improvements found during the security review.

A note about security audits

Inspired from Secureum:

"Audits are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible.

Audits can show the presence of vulnerabilities **but not their absence**."

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held understanding of the auditor's knowledge of security patterns as they relate to the client's contract(s), assuming that blockchain technologies, in particular, will continue to undergo frequent and ongoing development and therefore introduce unknown technical risks and flaws. The scope of the audit presented here is limited to the issues identified in the preliminary section and discussed in more detail in subsequent sections. The audit report does not address or provide opinions on any security aspects of the Solidity compiler, the tools used in the development of the contracts or the blockchain technologies themselves, or any issues not specifically addressed in this audit report.

The audit report makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, the legal framework for the business model, or any other statements about the suitability of the contracts for a particular purpose, or their bug-free status.

To the full extent permissible by applicable law, the auditors disclaim all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. The auditors hereby disclaim, and each client or user of this audit report hereby waives, releases and holds all auditors harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.

- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behaviour with some of the protocol's functionalities that's not so critical.

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions and the cost of the attack is relatively low to the amount of funds that can be stolen or lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a huge stake by the attacker with little or no incentive.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Low	Low	Low

Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.
- **Informational** - client **could** consider design/UX related decision
- **Recommendation** - client **could** have an internal team discussion on whether the recommendations provide any UX or security enhancement and if it is technically and economically feasible to implement the recommendations
- **Gas Findings** - client **could** consider implementing suggestions for better UX

Executive summary

Overview

Project Name	Brahma Finance: Brahma Console
Repository 1	https://github.com/brahma-fi/console-core/tree/ft-trusted-core
Commit hash 1	58bf05320bc5405f36549ca786a317724241e2ee
Repository 2	https://github.com/Brahma-fi/console-integrations/tree/ft-trusted-exec
Commit hash 2	630fdc4d3a21344c90c3fcda94ad045c6dbbe6dc
Documentation	Provided
Methods	Manual review

Issues found

Severity	Count
High risk	5
Medium risk	6
Low risk	6
Informational	8
Recommendations	13
Gas Findings	9

High Findings

[H-1] Loss of precision while calculating relative prices

Current Status

Enforcing order of operations **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L138
- https://github.com/Brahma-fi/console-integrations/blob/1fea2c1166ae2cedd74cdfa2986ff6d908fcda06/src/strategy/CoWDCAStrategy.sol#L273
- https://github.com/Brahma-fi/console-integrations/blob/1fea2c1166ae2cedd74cdfa2986ff6d908fcda06/src/strategy/CoWDCAStrategy.sol#L278

Description

While the logic to fetch the price of asset X in terms of Y is sound and would work perfectly for the scenario where the decimals of token Y is greater than decimals of asset X, however in the cases (mentioned in **Context**) where the decimals of asset X is greater than asset Y, there would be a definite loss of precision, especially in the absence of any scaling factor.

The loss in precision would become more pronounced with more differece in the decimals of X and Y (where decimals(X) >> decimals(Y)).

Recommended Mitigation Steps

Add a check to make sure that the decimals of asset Y is greater than or equal to decimals of asset X.

[H-2] USD Price Feeds with decimals different than 8 cannot be added

Current Status

Ackowledged, wont add tokens that dont follow 8 decimal price feed

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L63-L67

Description

The protocol erroneously assumes that any price feed with decimals other than 8 will not be a USD price feed. This is in line with the usual convention we see, however there are certain tokens with USD price feeds that have decimals different than 8.

Here are some examples:

1. AMPL/USD
    - Ampleforth
    - 0xe20CA8D7546932360e37E9D72c1a47334af57706
    - 18 decimals
    - Mainnet
2. XAU/USD
    - 0x7b219F57a8e9C7303204Af681e9fA69d17ef626f
    - 18 decimals
    - Goerli
3. AAPL/USD
    - Apple Stock
    - 0x4E4908dE170506b0795BE21bfb6e012770A635B1
    - 18 decimals
    - Avalanche
4. AMZN/USD
    - Amazon
5. SAVAX/USD
    - StakedAvax
    - 0x2854Ca10a54800e15A2a25cFa52567166434Ff0a
    - 18 decimals
    - Avalanche

These token feeds cannot be added to the protocol following the current logic.

Recommended Mitigation Steps

Make a list of all USD feeds that do not follow the 8 decimals convention and either add them manually by creating an exception for them or be ok with not allowing those feeds to be added to the protocol.

[H-3] No mechanism to handle revoking of price feeds

Current Status

In case oracle is experiencing DOS, try/catch is of no use as we would not do the execution either and revert anyways. We dont plan to implement a fallback solution using a twap from a different source as this is not a reliable for solution for all tokens, neither we can assume that a fallback will be available on chains other than mainnet

In case a price feed needs to be updated, governance can call setTokenPriceFeed to update to a new price feed

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L150-L151

Description

While currently there's no whitelisting mechanism to allow or disallow contracts from reading prices, powerful multisigs can tighten these access controls. In other words, the multisigs can immediately block access to price feeds at will.

Therefore, to prevent denial of service scenarios, it is recommended to query ChainLink price feeds using a defensive approach with Solidity's try/catch structure. In this way, if the call to the price feed fails, the caller contract is still in control and can handle any errors safely and explicitly.

For more information, refer this article from OZ.

Recommended Mitigation Steps

Wrap the `latestRoundData` function call in a try-catch block, and make arrangements in the catch block to replace the price feed address if and when required.

[H-4] Price Feeds with heartbeat greater than `DEFAULT_STALE_FEED_THRESHOLD` cannot be added

Current Status

Adding optional heartbeat [FIXED]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L28
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L84
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L175

Description

The value for `DEFAULT_STALE_FEED_THRESHOLD` constant is set as 90_000, which is equivalent to 25 hours and this value is passed as the `_staleFeedThreshold` in the function `_validateRound` for the

answers received after calling `latestRoundData` while trying to add a new price feed by calling `PriceFeedManager.setTokenPriceFeed`.

The function `_validateRound` basically ensures that no feed with a heartbeat more than the `DEFAULT_STALE_FEED_THRESHOLD` can be added to the PriceFeedManager.

Even though 25 hours is quite a long time to set as `DEFAULT_STALE_FEED_THRESHOLD`, there are feeds with heartbeat more than that.

Here is an example: AMPL/USD feed with a heartbeat of 48 hours.

According to the current protocol logic, there is no way to add this feed to the PriceFeedManager.

Recommended Mitigation Steps

Either increase the value of the constant `DEFAULT_STALE_FEED_THRESHOLD` or make it non-constant and include function to change it's value when trying to include feeds with large heartbeats.

[H-5] Upgrading a wallet may make the wallet useless

Current Status

replacing this with an option to deregister wallet [FIXED]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/WalletRegistry.sol#L118-L125
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/WalletRegistry.sol#L180-L182
- https://github.com/Brahma-fi/console-core/blob/ft-trusted-core/src/core/registries/WalletRegistry.sol#L143-L149

Description

Consider the `WalletRegistry.upgradeWalletType` function:

```
function upgradeWalletType() external {
    if (!isWallet(msg.sender)) revert WalletDoesntExist(msg.sender);

    uint8 fromWalletType = _walletDataMap[msg.sender].walletType;

    _setWalletType(msg.sender, _upgradablePaths[fromWalletType]);

    emit WalletUpgraded(msg.sender, fromWalletType,
_upgradablePaths[fromWalletType]);
}
```

In the function `WalletRegistry.upgradeWalletType`, let's look at line 4, here we cannot be sure that there exists a mapping for `fromWalletType` in the `_upgradablePaths` mapping.

This would set the **wallet type to 0**, when the mapping does not exist and then the wallet ceases to be a wallet (based on the function logic for function `WalletRegistry.isWallet`).

When it is not a wallet, you cannot do anything with that wallet, for example you cannot createSubscription with that wallet address. This makes the wallet useless and may lead to stuck funds for the users.

In such a case, the protocol will have no choice but to register the wallet again.

Other option could be that the protocol gov calls `setUpgradablePath` for 0, but that would cause conflicts with other users and it may very well NOT be allowed by the `WalletAdapterRegistry`.

Recommended Mitigation Steps

In the function `WalletRegistry.upgradeWalletType` add a check to ensure that the wallet type that the current wallet will be upgraded to, is supported.

Medium Findings

[M-1] Creation of task can fail if there are multiple authorized bots

Current Status

Governance can always remove bots if there occurs an instance of too many bots registered with BotManager

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BotManager.sol#L97-L109

Description

Creation of tasks in authorizedBots is done by `createTask` which iterates on all authorized bots and call `initTask` on bots. This function can suffer from Out Of Gas due to multiple bots and uncontrolled loop which can result in stoppage of Subscription creation.

Recommended Mitigation Steps

Make sure that authorized bots are not too much to avoid Out Of Gas exception. Alternative solution is to bound the for loop for certain highly available bots.

[M-2] Latest fixed compiler version can introduce permanent bug(s)

Current Status

We found that 0.8.18 and 0.8.19 fix some issues with abi.encodeCall which is being heavily used in the codebase. While we agree this concern is valid, we choose to with bug fixes that we know rather than

potential bugs which are unknown

Context

All the contracts which specify a compiler version, ie, `src/*.sol`.

Description

The current Brahma console contracts are not upgradable and if a critical bug in the compiler version 0.8.19 is discovered, the protocol cannot be updated to use a different version nor does it allow compilation via any other compiler version.

This essentially bricks the entire protocol, making it a sitting duck for hackers because even if the compiler bug is fixed in the next version, the protocol cannot use that version.

Please do not work with the assumption that nothing can realistically go wrong with the compiler because a lot of high and medium severity vulnerabilities have been found and patched in different versions of the solidity compiler throughout history. Here is a list of all previous bugs in Solidity compilers for your reference.

This risk is especially elevated when we are using the latest Solidity compiler because this particular version hasn't been battle-tested for a long time and active exploit research is still on going in this particular compiler version.

Recommended Mitigation Steps

1. Consider allowing 0.8.19 compiler version and higher by choosing to go with pragma solidity ^0.8.19.
2. Also, I deem the current compiler version to be too recent and therefore it is not time-tested. Consider switching to an earlier version. I recommend 0.8.13.

[M-3] WETH address can be misconfigured

Current Status

Add zero check for WETH, Cannot validate weth.name == “WETH” as this breaks compatibility with non ETH chains like Polygon, BSC, Gnosis etc [FIXED]

Context

- https://github.com/Brahma-fi/console-core/blob/main/src/core/PriceFeedManager.sol#L37-L39

Description

A zero address check is of **extreme importance** in this particular scenario, because if for any reason address(0) or any wrong address gets passed as the `WETH` address then there is no way to reset it.

Since, there is no way to reset/change it, the only plausible course of action would be to re-deploy the `PriceFeedManager` contract with the correct address which can use up significant time, enery and financial resources of the team.

Recommended Mitigation Steps

1. Introduce an address(0) check on the `_weth` address being passed.
2. Additionally, use another public function call to determine whether you passed the correct address or not. Something like: `_weth.name() == "Wrapped Ether"`

[M-4] Price freshness from Chainlink oracles is not fully validated

Current Status

added check [FIXED]

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L166-L178

Description

Chainlink oracle return values are not handled entirely, the priceFeed will return the following variables:

- roundId
- answer
- startedAt
- updatedAt
- answeredInRound

These return values are meant to be used to do some extra checks before updating the price. While almost all validations are done, the validation for `updatedAt` is not sufficient.

This could lead to stale prices according to the Chainlink documentation: https://docs.chain.link/data-feeds/price-feeds/historical-data

Recommended Mitigation Steps

Add another check which confirms that the `_lastUpdatedAt` value is non-zero.

```
require(_lastUpdatedAt != 0, "Latest round was not completed");
```

[M-5] Tokens with decimals >18 can get their price feed accepted

Current Status

Reverting in case this condition occurs [FIXED]

Context

https://github.com/Brahma-fi/console-core/blob/ft-trusted-core/src/core/PriceFeedManager.sol#L70-L76

Description

Consider the following code snippet from the function `PriceFeedManager.setTokenPriceFeed`:

```
if (token != ETH) {
    try IERC20Metadata(token).decimals() returns (uint8 _decimals) {
        if (_decimals > 18) revert InvalidERC20Decimals(token);

        tokenDecimals = _decimals;
    } catch {
        tokenDecimals = 18;
    }
} else {
    tokenDecimals = 18;
}
```

So, it is clear here that the protocol does not want to handle tokens with decimals more than 18, as generally, all reputed tokens have decimals 18 or below.

Therefore, ideally the protocol wants to reject all tokens with decimals > 18. However, if such a token happens to fail the call for `IERC20Metadata(token).decimals()`, then the `catch` block will be activated and it will go through while the protocol have erroneously assigned that token 18 decimals. They falsely assumed a token that will fail the decimals call will have 18 decimals exactly.

Recommended Mitigation Steps

Do not assume that a token that fails the `IERC20Metadata(token).decimals()` call, will have 18 decimals and simply revert.

[M-6] Protocol is not designed to handle *weird tokens*

Current Status

Acknowledged, Adding SOP off chain mechanisms to ensure no weird tokens are added

Context

- https://github.com/Brahma-fi/console-integrations/blob/1fea2c1166ae2cedd74cdfa2986ff6d908fcda06/src/automations/DCACoWAutomation.sol#L118
- https://github.com/Brahma-fi/console-integrations/blob/1fea2c1166ae2cedd74cdfa2986ff6d908fcda06/src/strategy/CoWDCAStrategy.sol#L128-L136

Description

Please go through the list of weird tokens to understand what are weird tokens and in which sense are they weird.

The protocol in general is not designed to handle such weird tokens.

For example, let's consider the case of pausable tokens or tokens that can blacklist a user.

So, suppose a user discovers that they have been blacklisted by USDC all of a sudden and then they want to exit from their current strategy.

When the user calls the `exitStrategy` function, their `tokenIn` might be USDC which might have blocklisted them, but their `tokenOut` might be something else, which they should be perfectly eligible to get. However, in the case of `tokenIn` transfer to the user failing, the user would not be able to get even their `tokenOut`, as the entire transaction will revert.

This is just one of many examples, and such scenarios can be observed at many other places such as while the transfer of fees, if a token is such that it takes a fee for transfers, then the fee transfer would always fail.

Recommended Mitigation Steps

1. Be very careful around which tokens the protocol allows to be whitelisted
2. Introduce checks and fail-safe mechanisms to be able to handle the worst case scenario arising from the whitelisted tokens, such as a user getting blocklisted by USDC.

Low Findings

[L-1] `PriceFeeManager` doesn't check if Arbitrum (other L2) sequencer is down in Chainlink feeds

Current Status

Acknowledged.

Context

- https://github.com/Brahma-fi/console-core/blob/main/src/core/PriceFeedManager.sol#L147-L156

Reason for Low Severity

- For certain types of protocol, like, lending protocols this would have been a high severity finding.
    - Because, shutting down of the sequencer does not mean that the L2 chain has stopped working
    - People can still submit their transactions and interact with the chain using the L1 contracts (force inclusion mechanism)
    - However, not everyone might have the expertise to be able to interact with the L2 chain via the L1 contracts
    - This could mean, in the time when the sequencer is down, if the price of the borrower's collateral goes down and a lender with their knowledge of using `forced inclusion` may give the borrower a margin call.
    - The borrower might not have the technical expertise to respond to that margin call via the L1 contracts and end up getting liquidated
- Therefore, this check is essential when looking to provide equal opportunities and access to all the participants in your protocol.

- In this particular protocol, the impact is limited to the possibility that the fee might be calculated using an earlier price and the actual transaction might cost something else.
- Therefore, we have decided to categorise this *issue* as a low-severity issue.

Description

Optimistic rollup protocols transfer all execution from the Layer 1 (L1) Ethereum chain to a Layer 2 (L2) chain, perform the execution on the L2 chain, and then return the results to the L1 chain.

These protocols employ a sequencer to process and aggregate L2 transactions by consolidating multiple transactions into a single one.

If a sequencer becomes unavailable, access to read/write APIs that consumers rely on becomes impossible, causing applications on the L2 network inaccessible for most users, unless they interact directly through the L1 optimistic rollup contracts.

While the L2 chain has not stopped, it would be inequitable to continue offering service on your applications when only a select few can utilize them.

Recommended Mitigation Steps

1. A straightforward fix exists. Use the sequencer uptime feed to monitor the sequencer's online status and prevent consumption of the price feed when the sequencer is offline.

2. Here's a sample implementation to give an idea of how to use the sequencer feed:

```
constructor() {
    sequencerUptimeFeed = AggregatorV2V3Interface(someAddress);
}

function getLatestPrice() public view returns (int) {
    (
        ,
        int256 answer,
        uint256 startedAt,
        ,

    ) = sequencerUptimeFeed.latestRoundData;

    bool isSequencerUp = answer == 0;
    if(!isSequencerUp) {
        revert SequencerDown();
    }

    uint256 timeSinceUp = block.timestamp - startedAt;
    if(timeSinceUp <= GRACE_PERIOD_TIME) {
        revert GracePeriodNotOver();
    }

    (
        ,
        int price,
```

```
            ,
            ,

        ) = priceFeed.latestRoundData();

        return price;
    }
```

[L-2] Low level calls can result in Out Of Gas if the call returns large returndata

Current Status

Acknowledged

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L214
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L267
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L98

Reason for low severity

- In and of itself this would have been a high severity issue because for a protocol to be so susceptible to a DoS attack is a high severity issue.
- However, we understand that the governance of the protocol would prevent any untrusted actors from being able to access this function and therefore this reduces the *likelihood* of such an attack happening
- However, the protocol governance must be vigilant while adding new strategies owing to this attack vector.

Description

The above low level calls call the recipient using CALL or DELEGATECALL opcode and copies the return data from the recipient of the call into memory. It is possible that the recipient can return huge return data which can cause memory expansion in caller contract resulting in consumption of all the gas.

Recommended Mitigation Steps

Perform a call in assembly block and only copy the return data needed.

[L-3] Unchecked targets for low-level calls may promote false positives

Current Status

Implementing adddress ≠ 0 [FIXED]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/wallets/SafeWalletAdapter.sol#L40
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/TokenTransfer.sol#L33
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/TokenTransfer.sol#L21
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L98
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L267

Reason for low severity

- In and of itself this would have been a high severity issue because a false positive resulting from a call returning success for a call that did not actually succeed can create quite critical issues.
- However, we understand that the targets are being meticulously identified by the protocol and the protocol-whitelisted strategies and therefore a low-level call happening to an address that does not exist is quite low, but still a possibility.
- Therefore, since this issue has a high impact but a very low probability of occuring, it has been categorised as a low-severity issue.

Description

Low-level Solidity calls such as `staticcall`, `call` or `delegatecall` return a boolean and a bytes parameter when they are called on a particular address.

The bool represents whether the call was successful or not.

The caveat however, is that if any of this low-level call is made on `address(0)` or any non-existent address, the boolean returned would still be `true`.

Therefore, this can trick the protocol into thinking that a particular low-level call was successful, while in reality the `call` was not even made to the intended address.

Recommended Mitigation Steps

1. Introduce a check on the call `target` to check it is not `address(0)`.
2. Couple that with a check to make sure that the `target` is indeed a contract, by using isContract at relevant places.

[L-4] Missing address checks can have negative consequences

Current Status

Acknowledged and Fixed.

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/AddressProvider.sol#L65
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/FeePayer.sol#L41
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/AddressProvider.sol#L80
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/bots/GelatoBot.sol#L28-L33

Description

For setting values of critical external contracts, consider using atleast a few checks on the address being passed. Given, the non-upgradable nature of `Brahma-console` contracts, it is essential to ensure that the correct addresses are being passed because the only recourse in event of setting wrong address would be redeployment (in most cases), which is a less than ideal solution.

This tips mentioned in the *Recommended Mitigation Steps* is in general applicable for all contract value setting and not restricted to the specific ones mentioned in *Context*.

This issue has the same reasoning as to why `_supportsAddressProvider` is being to ascertain that a particular address can support `AddressProvider` type contracts.

Recommended Mitigation Steps

1. At the very least consider adding `address(0)` and `isContract` checks to ensure that the address being passed has a high probability of being the correct one.
2. Consider calling a public getter function once set and compare the value returned to the expected value for more robustness.
3. Consider using standards like ERC165 and/or ERC1820, to ensure that the correct address type is being used.

[L-5] It might be possible to register wallet adapter for an invalid wallet type

Current Status

Added the check **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/WalletAdapterRegistry.sol#L26-L32
- https://github.com/Brahma-fi/console-core/blob/ft-trusted-core/src/core/registries/WalletAdapterRegistry.sol#L44-L46
- https://github.com/Brahma-fi/console-core/blob/ft-trusted-core/src/core/registries/WalletRegistry.sol#L143-L149

Reason for low severity

- The governance can call this function again to reset the wallet adapter of the invalid type of wallet back to `address(0)`.
- Therefore, this would be a reversible change/issue.

Description

Consider the function `registerWalletAdapter` for reference:

```
function registerWalletAdapter(address _adapter) external {
    _onlyGov();

    uint8 _walletId = IWalletAdapter(_adapter).id();
    _setWalletAdapter(_walletId, _adapter);

    emit WalletAdapterRegistered(_adapter, _walletId);
}
```

In this function we see that it is possible for a wrong `_adapter` address or a malicious `_adapter` address to return 0 in line 4. This way, we will set and potentially reset the adapterAddress for walletId = 0.

Ideally for a wallet with ID = 0, there should not be a adapter address as we know that a wallet with walletId = 0 is not a wallet. This is because of the function `WalletRegistry.isWallet` logic. Here is the `isWallet` function for reference:

```
function isWallet(address _wallet) public view returns (bool) {
    WalletData memory walletData = _walletDataMap[_wallet];
    if (walletData.walletType == 0 || walletData.feeToken == address(0)) {
        return false;
    }
    return true;
}
```

Recommended Mitigation Steps

- Add a check in the `registerWalletAdapter` function that ensures that the ID received

[L-6] All checks for `DCACoWAutomation.canInitSwap` not implemented

Current Status

The check existed before we decided not to store existing order id as it served no practical purpose and allows to save gas considerable gas Updated natspec **[FIXED]**

Context

- https://github.com/Brahma-fi/console-integrations/blob/1fea2c1166ae2cedd74cdfa2986ff6d908fcda06/src/automations/DCACoWAutomati

on.sol#L42-L52

Description

The natspec comments for the `DCACoWAutomation.canInitSwap` function lists out 4 different checks that should be implemented in the function to determine whether the swap can be initiated or not.

Here are all the checks that were listed:

- Checks if a DCA swap can be executed
- Checks for existing active orders
- Checks if subAccount has enough balance to execute swap
- Checks if enough time has passed since last swap

All checks have not been implemented.

Recommended Mitigation Steps

Implement all the checks listed out in the function comments.

Informational Findings

[I-1] Wrong naming of event emitted

Current Status

Updated event **[FIXED]**

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/SafeDeployer.sol#L155

Description

If there is already available subaccounts, allocateOrDeployFreshSubAccount will allocate the available subaccount and won't deploy new one. Even if this is the case, event named `subAccountDeployed` will be emitted which shouldn't be because no subAccount got deployed.

Recommended Mitigation Steps

Modify the naming of event.

[I-2] Wrong natspec

Current Status

Updated natspec **[FIXED]**

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BotManager.sol#L77-L88

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/FeePayer.sol#L70-L74

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/wallets/SafeWalletAdapter.sol#L74-L78

Description

Natspec documentation is useful for internal developers that need to work on the project, external developers that need to integrate with the project, auditors that have to review it but also for end users given that. The above snippet linked misses the spec for param `externalTask`. There has been several instances like this in whole project. One such example is: https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BotManager.sol#L69-L72

Recommended Mitigation Steps

Correct the natspec.

[I-3] Function `hasZeroBalance` can be simplified

Current Status

Updated method **[FIXED]**

Context

https://github.com/Brahma-fi/console-integrations/blob/ft-trusted-exec/src/automations/DCACoWAutomation.sol#L60-L63

Description

Function `hasZeroBalance` can be simplified and can be written without using `if` statement. Similar changes can be done in `hasNonZeroBalance` and `isWalletTypeSupported`.

Recommended Mitigation Steps

Change the `if` statement to `return IERC20(token).balanceOf(owner) == 0` for `hasZeroBalance`.

[I-4] Sequence of events for `createSubscription` in natspec is different from what is shown in architecture diagram.

Current Status

[FIXED]

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/SubscriptionRegistry.sol#L57-L61

Description

Steps shown in subscription flow is different from what is shown in architecture diagram.

Recommended Mitigation Steps

Rectify architecture diagram.

[I-5] function `_packMultisendTxns` can result in Out Of Gas exception.

Current Status

It would be better to let it reach out of gas state rather than actively reverting as the outcome is same

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/SafeHelper.sol#L64

Description

Function mentioned can result in Out Of Gas exception for multiple transactions.

Recommended Mitigation Steps

Set an upper limit for number of transaction to multi send.

[I-6] Error in `_packMultisendTxns` can be made more informative

Current Status

Added index **[FIXED]**

Context

https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/SafeHelper.sol#L75

Description

The function on revert doesn't state which transaction made it to revert.

Recommended Mitigation Steps

Try to include the index of transaction which made the whole batch to fail.

[I-7] No mechanism to reliably know whether AddressProvider is ready to provide all addresses

Current Status

Added check **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/SafeDeployer.sol#L195-L197
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/bots/BaseBot.sol#L20
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/bots/GelatoBot.sol#L120
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/AddressProviderService.sol#L27-L31
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Authorizer.sol#L33-L38

Description

While this is a sound mechanism to fetch different addresses across the protocol by centralising the book-keeping of addresses in `AddressProvider.sol`, we must note carefully that the `Brahma-console` contracts are not upgradable and if in any contract `addressProvider` is used to fetch an address **before** that address was initialized in the `AddressProvider` contract itself, it can permanently set those addresses to `address(0)`.

Recommended Mitigation Steps

Come up with a mechanism to ensure that the `AddressProvider` contract is deployed and fully initialized.

For example, consider the following code snippet from the `AddressProviderService` contract:

```
    constructor(address _addressProvider) {
        if (_addressProvider == address(0)) revert
InvalidAddressProvider();
        addressProvider = AddressProvider(_addressProvider);
        strategyRegistry = addressProvider.strategyRegistry();
        subscriptionRegistry = addressProvider.subscriptionRegistry();
        subAccountRegistry = addressProvider.subAccountRegistry();
        walletAdapterRegistry = addressProvider.walletAdapterRegistry();
        walletRegistry = addressProvider.walletRegistry();
    }
```

Now, here how do you ensure that the values in `AddressProvider` contract are set and the values queried here won't return `address(0)`, but some legit values.

One solution could be to introduce a function in AddressProvider.sol that returns a bool on whether all these relevant values are set or not

Basically, we need to work on a mechanism to ensure that all the variables of the `AddressProvider` contract has been initialized as expected before that contract starts getting used to fetch different addresses.

[I-8] Unnecessary parameters added to `_generateSingleThresholdSignature`

Current Status

Removed unused parameters **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/SafeHelper.sol#L42-L57

Description

In the current form, the function `_generateSingleThresholdSignature` works just as expected, however this could be made more efficient according to the official Gnosis documentations.

Recommended Mitigation Steps

1. Do away with the last two parameters passed, which are extra and not really required.
2. Since the extra parameters has been removed, you can also redo the third parameter which basically points to the location of the extra parameter.

Recommendations

[R-1]

Current Status

[Noted]

Description

User doesn't have the ability to set the slippage parameters on their own. The user is **forced** to do the transaction with the slippage set by governance for the strategy. It's better to take parameter slippage from the user where they decide maxSlippage based on their risk tolerance.

[R-2]

Current Status

[Noted]

Description

While running the function `_packMultisendTxns` with a high number of transactions, say, 10,000 `_txns`, if the last `_txnsCallType` is `StaticCall`, then the entire function will revert. Meaning that all the gas spent processing the previous 9,999 transactions goes to waste. Consider implementing a try-catch block and note which particular transactions failed to try them again.

[R-3]

Current Status

[FIXED]

Description

Try to use allowlist and blocklist instead of whitelist and blacklist for feeTokens.

[R-4]

Current Status

[Thanks, Noted]

Description

Consider using this snippet to detect any change in proxies, since it is understood that the protocol does not want to include strategies that can be upgraded.

[R-5]

Current Status

[FIXED]

Description

Considering the fact that the Brahma contracts are not upgradable, it would be better to not make the variables `GAS_OVERHEAD_NATIVE` and `GAS_OVERHEAD_ERC20` as constant and introduce functions to change their values in the future if and when required.

[R-6]

Current Status

[FIXED]

Description

Usage of `abi.encode` instead of `abi.encodePacked` in `require` strings in `Executor.sol` will result in easier decoding.

[R-7]

Current Status

[Noted]

Description

Consider adding a check on to make sure that all the addresses being set in the `CoreAuth` constructor are different.

[R-8]

Current Status

Enforcing order of operations **[Noted]**

Description

Consider adding a time delay in `acceptGovernance` when governance is changed.

[R-9]

Current Status

[FIXED]

Description

Consider replacing `common` with `general` in `Authorizer.sol` contract description.

[R-10]

Current Status

Noted, Any recommendations on backup oracles?

Description

Consider adding a backup oracle. Regarding any price spikes it is straightforward to construct a mitigation mechanics for such cases, so the system will be affected by sustainable price movements only.

As price outrages provide a substantial attack surface for the project it's worth adding some complexity to the implementation. One of the approaches is to track both current and TWAP prices, and condition all state changing actions, including liquidations, on the current price being within a threshold of the TWAP one.

If the liquidation margin level is conservative enough and TWAP window is small enough this is safe for the overall stability of the system, while providing substantial mitigation mechanics by allowing state changes on the locally calm market only.

[R-11]

Current Status

`canExecute` check is already present

Description

In the function `BrahRouter.executeAutomationViaBot`, consider adding the following two checks:

- add a call for canExecute here before calling `executeAutomation`
- put a check here to see if `_wallet` is the owner of `_subAccount`

[R-12]

Current Status

Enforcing order of operations **[Noted]**

Description

For functions like `BrahRouter._executeSafeERC20Transfer`, be **extremely clear and vigilant** on the decoding logic, because this is multi-step decoding process for the returning bytes values. Try and test these type of functions **heavily**, much more comprehensively than the current testing done for such functions.

[R-13]

Current Status

Yes, intentional, only large cap tokens planned to be supported as fee tokens, if something happens, users will be notified to change their fee tokens

Description

The `WalletRegistry` contract implements a function called `addWhitelistedFeeToken` which is the function which allows for whitelisting of tokens to be used as fee tokens in the protocol.

However, there is no function to remove a whitelisted token.

This seems like an intentional step from the team, but we would still advice the team to form a solid procedure to be implemented in case of any eventualities such as de-pegging, user blacklisting, price feed deprecation of the whitelisted tokens.

Gas Findings

[G-1] Explicit setting of local variable to 0 is not needed.

Current Status

[Acknowledged]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/SubscriptionRegistry.sol#L90
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/SubscriptionRegistry.sol#L137
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/SubscriptionRegistry.sol#L217
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BotManager.sol#L100
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BotManager.sol#L123
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BrahRouter.sol#L183
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L133

[G-2] Packing of struct can be done to reduce from 3 slots to 2 slots

Current Status

Added the refactor to a single slot **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#L21-L25

[G-3] Function signature of `setTokenPriceFeedStaleThreshold` can be changed to take array of `tokenAddress` and `threshold` to set them all at once. Similar things can be done in `updateSubData` because strategy needs to call this function individually for all subAccount.

Current Status

[Acknowledged]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/PriceFeedManager.sol#LL101C21-L101C21
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/SubscriptionRegistry.sol#L155

[G-4] In function `_packMultisendTxns`, variables are declared inside of the do-while loop. The same optimization can be done in `requestSubAccount`.

Current Status

[Acknowledged]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/SafeHelper.sol#L71-L78
- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/BrahRouter.sol#L186

[G-5] In function `_packMultisendTxns`, put the if condition in else case since the more likely condition should be in if block.

Current Status

True that will be most likely condition but the other condition will be more frequent in case of long multisend call

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/libraries/SafeHelper.sol#L84-L90

[G-6] Early check on whether the _feeToken is whitelisted or not, will save a lot of gas since that check is made in the last step of function `deployConsoleAccount`.

Current Status

Thanks, **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/SafeDeployer.sol#L61

[G-7] Use custom errors instead of require.

Current Status

Thanks, **[FIXED]**

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/Executor.sol#L215

[G-8] Consider passing array of `RegisterKey` of length n and containing n addresses, instead of calling `initRegistry` n times.

Current Status

[Noted]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/AddressProvider.sol#L131-L153

[G-9] Adding a condition `_tokenRequests.length > 0` before the execution of `brahRouter.requestSubAccountFunds` in `SubscriptionRegistry.createSubscription` may lead to gas savings.

Current Status

[Noted]

Context

- https://github.com/Brahma-fi/console-core/blob/58bf05320bc5405f36549ca786a317724241e2ee/src/core/registries/SubscriptionRegistry.sol#L109