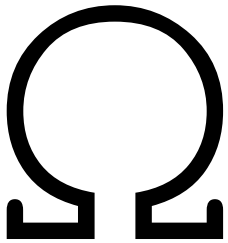

Team Omega

https://teamomega.eth.limo

Giza Pendle Audit

Final Report

August 16, 2025

Summary **3**
Scope of the audit **4**
Methodology 4
Liability 5
Severity definitions **5**
Summary of findings **5**
Resolution **6**
Findings **6**
General 6
G1. Unaddressed issues from older reports [medium] [resolved*] 6
G2. Error handling skips silently over errors [high] [resolved] 7
G3. Gas prices are not taken into account [medium] [resolved] 9
G4. Incomplete logic for multiple deposits [low] [resolved] 9
G5. Incomplete top-up logic [low] [resolved] 9
Pyproject.toml 10
PP. Dependency versions are not pinned [low] [will be resolved] 10
api/deps.py 10
DEP1. Mainnet and Sonic networks missing from get_business_adapter [medium] [resolved*] 10
api/adapters/business_adapter.py 10
ABA1. The value for amount in activate_wallet() is never checked [medium] [resolved] 10
ABA2. Run may be triggered multiple times on wallet activation [low] [not resolved] 11
ABA3. Run endpoint allows execution in states where wallet shouldn’t run [low] [not resolved] 11
entities/wallet.py 12
W1. Setter for selected_chains does not work [medium] [resolved] 12
W2. Unused variables and functions [info] [resolved] 12
api/v1/endpoints/proxy.py 12
P1. Authentication does not verify correct user address [low] [resolved] 12
business/impl/bridge_estimator.py 13
BE1. Missing verification for price returned from API [medium] [resolved] 13
business/impl/planner.py 14
PL1. Wrong input value in swap action [medium] [resolved] 14
business/impl/db/sql_writer.py 15
SQLW1. Missing activation_date on wallet update [low] [resolved] 15
SQLW2. Unused functions [info] [resolved] 15
business/impl/db/sql_reader.py 15
SQLR1. Check current TVL can be manipulated [medium] [not resolved] 15
SQLR2. Unused functions [info] [resolved] 16
business/impl/manager.py 16
M1. Wrong check for sufficient delta APR on run [low] [resolved] 16

M2. Funds may get stuck on a low yield protocol due to reallocation constraint [low] [not resolved] 16
business/impl/constants.py 17
C1. Missing mainnet values [medium] [resolved] 17
business/impl/optimizer.py 17
O1. Market names are treated as unique while not necessarily being unique [high][resolved] 17
O2. Error in setting up constraints for market_apys [high] [resolved*] 18
O3. _get_effective_apy_for_funding does not take into account that funds may be allocated from other chains [medium] [resolved*] 19
O4. Bridging costs are not correctly treated in _get_effective_apy_for_funding [low] [resolved] 19
O5. Failing to apply a constraint is silently ignored [low] [resolved] 20
O6. Wrong value calculated in initial weighted APY [low] [resolved] 20
O7. MAX_FEE may underestimate costs [low] [resolved] 21
O8. Piece calculation may be generated even when getting APY fails [low] [not resolved] 21
O9. Swap data result should use effectiveAPY instead of impliedAPY [low] [resolved] 22
job/pendle_job/main.py 22
JM1. Using of non-existent DEFAULT_THRESHOLD [low] [resolved] 22
pendle/business/impl/tools/* 22
T1. Post execution of tools may save empty tx hash [low] [resolved] 22
T2. Code duplication between tools [info] [partially resolved] 23
Arma Contract Interactions 23
packages/arma_chain/src/arma_chain/chain_ops/enums.py 23
CE1. Missing BRIDGE action [info] [resolved] 23
packages/arma_chain/src/arma_chain/chain_ops/tx_manager.py 23
TXM1. Unnecessary approve call on transfer [low] [not resolved] 23
packages/cross_chain/src/cross_chain/pendle/pendle.py 24
CPP1. Slippage is not being validated [high] [resolved] 24

Summary

Team Omega
Team Omega (https://teamomega.eth.limo/) specializes in Smart Contract security audits on the Ethereum ecosystem.

Giza

Giza.tech (https://www.gizatech.xyz/) is developing a platform for AI solutions in web3

The system we audited is an agent that invests ETH in Pendle instances across various chains.

A short specification document was shared with us at https://docs.google.com/document/d/1uqMcd9JaZ78JZnuVCBgmE4Q5Nu2QijQntUMA2bMGBSA/

Scope of the audit

We audited the code in the following repositories, which together define a system for allocating investments in yield-bearing defi products.

Specifically, Team Omega will audit the code in the following two repositories:

1. `https://github.com/gizatechxyz/pendl-agent/`
2. `https://github.com/gizatechxyz/arma-contract-in`teractions

The code in scope is:

1. Regarding the `pendl-agent` repository: the python code in the job and the job directories at commit `54fde69fa4f062986326ed495c3407e727c1e201`

These are about 6000 normalised lines of Python code (excluding blank lines and comments)

2. Regarding th`e arma-contract-interactions` repository: the scope is the Python files in the following directories:
    - `packages/arma_chain/src`
    - `packages/base_chain/src`
    - `packages/cross_chain/src/`

limited to the diff between commit `81ccd3bc9079a7f2630149adbb635d1d422c0fe2`, and the code we audited in our last audit report (https://docs.google.com/document/d/1S1KC3Enij9pyqXYgPycFxEvpQHIus9kpIYyw9zTOtFY/) , which was this one `731405cfecee5a89b3c33e2eec0c6ddae94cff5a`,

These are about 1800 normalized lines of Python code

Methodology

The audit report has been compiled on the basis of the findings from different auditors (Jelle and Ben). The auditors work independently. Each of the auditors has several years of experience in developing smart contracts and web3 applications.

Liability

The audit is on a best-effort basis. In particular, the audit does not represent any guarantee that the software is free of bugs or other issues, nor is it an endorsement by Team Omega of any of the functionality implemented by the software.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Summary of findings

We found **4** high severity issues - these are issues that can lead to a loss of funds, and are essential to fix. We classified 10 issues as “medium” - these are issues we believe you should definitely address. In addition, 18 issues were classified as “low”.

As this review focused on “security issues” - in this case, on bugs that may lead to loss of funds or to inefficient allocations of funds, we only include some very generic issues of the category “info”. As a general note, we would like to recommend removing unused code, and add more comments where appropriate.

Severity	**Number of issues**	**Number of resolved issues**
High	4	4
Medium	10	10

Low	18	11

Resolution

The Giza team presented us with the following commit, in which all issues of high and medium severity were fixed.

For the pendl-agent:

`6e0312f3879cd06040410fdc83cd5771ca553945`

For arma-contract-interactions:

`be06355bbd0e374c782a6a9d67b8412efc7b353a`

We reviewed the fixes and added a “resolution” section to each of the issues.

A number of issues are related to the fact that the software operates among different chains (namely G1, DEP1, O2 and O3). The team communicated that the first release of the bot will only operate on Base, so these issues are not currently relevant. We marked these issues as “[resolved*]”.

Findings

General

G1. Unaddressed issues from older reports [medium] [resolved*]

Although not strictly in scope, it is important to remind the developers that some issues from previous audit reports were not addressed in this release. Specifically, we mention:

- Issue C3 in 202505-Giza Tech ARMA Audit II -Final Report (gas limit logic could lead to failed transactions (low))
- Issue M1 in 202505-Giza Tech ARMA Audit II -Final Report (jobs API key can access any endpoint of any wallet (medium))

Also issue A2 from 202505-Giza Tech ARMA May Audit IIII -Final Report (Top-up logic is confusing and can be abused (medium)) which was resolved, re-appears in the current code base.
Resolution: C3 was not addressed in the code, but the transactions are sent to an endpoint that ignores the “gas” parameters, so it’s impact is limited. M1 was not addressed either, but is out of scope of this report. Finally, issue A2 was addressed by removing the “top_up” endpoint.

G2. Error handling skips silently over errors [high] [resolved]

In many places through the code base, errors pertaining to failures of external services, such as database read/write errors and errors deriving from requests to external APIs, are passed over silently. Often, a value is then returned that is not distinguishable from a “good” value (such as an empty list when a query fails). These values are then handled upstream as “good” values, and this may lead to unwanted results.

We list here a number of places where this is particularly clear, but this list is probably not complete.

1. In `data.py`, several functions, which is responsible for sending http requests to the pendle API, functions that fail to retrieve data will return default values when an error occurs:

Method	Error Handling
`_get_active_markets_by_chain`	Logs error, returns `[]`
`_get_asset_prices`	Logs error, returns `{}`
`get_historical_data`	Logs error, returns `None`
`get_swap_data`	Logs error, returns `None`
`is_market_expiring`	Logs warning, returns `False`
`get_market_apr`	Logs warning, returns `0.0`

`get_market_address_by_token`	Logs warning, returns `None`
`_get_market_by_token`	Logs warning, returns `None`

This pattern is dangerous, as many of these values are used upstream as the good values. For example, if `_get_active_markets_by_chain` returns an empty list because the API is slow to respond, the markets from that chain will not be considered in the optimization problem - both in calculating the value of the portfolio to optimize, nor as possible allocation destinations, which can lead to suboptimal results.

2. Similarly, in `sql_reader.py`, which is responsible for reading data from the SQL database, the logic systematically skips over database `ValidationError`, and returns a default value if such an error occurs. The following functions are affected

Function Name	Error Handling Style
`get_wallet_information`	Logs `ValidationError,` returns `None`
`get_performance_chart_data`	Logs `ValidationError` in loop, omits data from result
`get_transaction_history`	logs `ValidationError,` omits data from result
`get_tos_sign_data`	Logs `ValidationError` and `TypeError`
`check_current_tvl`	Logs *any* exception, returns 0
`_get_wallets`	logs `ValidationError,` omits failing wallet from return value
`get_transaction`	logs `ValidationError`

`get_wallet_by_telegram_id`	logs `ValidationError`

This is problematic for two reasons. The first reason is similar to the one we described before: in case of an error, the return values are used upstream as “good” values. The second reason is that the `ValidationErrors` can point to a systematic failure in writing data to the database correctly. If that is the case (i.e. if the code is writing invalid data to the database) this is something to be resolved, not simply logged.

3. This kind of leaky error handling is also done in `optimizer.py` For example, if an `expiry_data` cannot be parsed in `_calculate_bridge_impact_annual`, a default value of `1` is used silently (practically inflating the bridge costs so that this path is excluded). Similar patterns are used in `_precompute_bridge_rates`, in `_prepare_optimization_data`, and in `_get_effective_apy_for_funding`, which contain essential logic for a successful run of the optimization problem, but skip over *any* error. This way of handling errors typically excludes entire markets from consideration, which would typically represent an opportunity cost, but can also lead to loss of funds. See also issue O8 below.

4. In `check_to_execute` in `data_retriever.py`, which checks if a job should be executed, *any error* is logged but skipped over, and the function returns `True` (i.e. the job is considered safe to execute, even if the constraints are n met)

Recommendation: Raise errors where appropriate instead of only logging them. If this a problem for reasons of liveness or otherwise, at least return values that are distinguishable from “good” values (.e.g. `None` instead of an empty list), but take care to handle these error values properly upstream.
Resolution: Error handling has been greatly improved throughout the code, and the examples we mentioned in this issue were addressed.

G3. Gas prices are not taken into account [medium] [resolved]

Depending on the chain, on the amount invested, and the frequency of operations, gas prices for withdrawal, swapping and depositing can negatively influence the APR of an investment opportunity considerably, yet they are not taken into account for the decision making.
Recommendation: Take gas prices into account.
Resolution: The developers indicated that this is intentional, and gas costs are sponsored.

G4. Incomplete logic for multiple deposits [low] [resolved]

The current logic of the code allows only for one deposit to be saved for a user. However other parts of the code, such as the APR logic, assumes multiple deposits are saved each time the

user added funds. This means certain APR calculations may not work as expected. Also in general, the top up logic itself seems wrong and incomplete.
Recommendation: Save each new deposit without overriding the older ones.
Resolution: The deposit logic was overhauled and this issue is not relevant anymore.

G5. Incomplete top-up logic [low] [resolved]

The current top-up logic only calls the wallet `run` function, but it does not use the `amount` value of the new deposit passed to it nor updates the deposits list in any way. Also, the `top_up` function in the `manager.py` file is both unused, and uses a function incorrectly implemented (`register_new_deposit` which always sets the deposit token to USDC address on Base regardless of the deposit chain).
Recommendation: Either fix or disable the top-up logic.
Resolution: The top-up endpoint was removed.

Pyproject.toml

PP. Dependency versions are not pinned [low] [will be resolved]

In `pyproject.toml`, the dependencies are not pinned to specific versions. This makes the exact builds unpredictable, and makes it easier for compromised new package versions to be included in the build.
Recommendation: Use precise versions of dependencies.
Resolution: The team has indicated that it will pin the versions before releasing the code

api/deps.py

DEP1. Mainnet and Sonic networks missing from get_business_adapter [medium] [resolved*]

The `get_business_adapter` function is responsible for creating an instance of the business adapter. The default for these does not include Mainnet and Sonic networks. It can receive a list of `chain_ids`, but those are not passed in any of the places where the function is called in the code base under review. This means these networks will not be added to the business adapter and be missed by the system.
Recommendation: Add all networks to the default chain_ids of `get_business_adapter`.
Resolution: The team stated that at launch, these networks are not supported.

api/adapters/business_adapter.py

ABA1. The value for amount in activate_wallet() is never checked [medium] [resolved]

The function `activate_wallet()` takes a value for amount that is passed on by the caller in the request. The value will be used for the `Deposit` record in the database, and is written when activating the wallet for the first time, but also when reactivating a wallet.
This value is never verified against any actual on-chain deposit. It is also not clear on which chain this amount would be deposited. As the `Deposit` value is used in various parts of the agent logic, this represents a potential problem.
Recommendation: It is not completely obvious how this problem should be resolved, but the value should be validated, or perhaps be read, from on-chain data.
Resolution: The value for the first Deposit is now simply set to the current token balance.

ABA2. Run may be triggered multiple times on wallet activation [low] [not resolved]

The `activate_wallet` function checks the current status of the wallet, and in case it is `ACTIVATING`, the `run` command for the wallet will be triggered. This means if the user calls the activation endpoint multiple times while it's still running (for example accidentally clicking a button multiple times), the wallet `run` logic will be triggered multiple times unnecessarily.
Recommendation: In case the wallet status is `ACTIVATING` just return without executing more of the code.
Resolution: The issue was not resolved. The team acknowledged the issue and will address it in a future iteration.

ABA3. Run endpoint allows execution in states where wallet shouldn't run [low] [not resolved]

The `run` endpoint verifies the wallet status is not `DEACTIVATED`, `EMERGENCY`, or `DEACTIVATING`, and will continue with the `run` operation if it isn't. Yet there can be more states where running the wallet is not desirable, like `ACTIVATING`, `RUNNING`, `DEACTIVATION_FAILED`, and likely any state which is not `ACTIVATED`.
Recommendation: Check if wallet status is activated and cancel the execution if it isn't.
Resolution: The issue was not resolved. The Giae Team acknowledged the issue and stated that "We acknowledge the recommendation. However, the wallet may temporarily be in non-ACTIVATED states such as ACTIVATING, RUNNING, or DEACTIVATION_FAILED due to delays or failures from external providers. In such cases, the run_wallet job should still be allowed to execute to ensure that the agent can recover, progress its state, or complete pending

operations. Restricting execution strictly to the ACTIVATED state could cause unnecessary interruptions or leave the wallet stuck in a non-recoverable state.”

entities/wallet.py

W1. Setter for selected_chains does not work [medium] [resolved]

The setter function for `selected_chains` calls the `_mark_dirty` with the `selected_protocols` key instead of the `selected_chains` key. This means that the new value for `selected_chains` is never saved when trying to update it.
Recommendation: Mark the `selected_chains` instead of `selected_protocols` as dirty
Resolution: The issue was resolved as recommended.

W2. Unused variables and functions [info] [resolved]

The `WETH_ADDRESS`, `USDC_ADDRESS`, `ETH_ADDRESS`, `get_balance`, `add_transaction`, `add`, `activate`, `register_new_deposit` functions and variables are unused, and some also contain certain mistakes/ lack implementation.
Recommendation: Remove unused code and avoid leaving code which is not properly implemented to avoid mistakenly calling it in the future.
Resolution: The issue was resolved as recommended. The functions were removed or fixed and put into use.

api/v1/endpoints/proxy.py

P1. Authentication does not verify correct user address [low] [resolved]

The `proxy_post` function is protected with authentication to ensure the user who called the endpoint cannot access wallets of other users. In the authentication process, the code allows for two different fields for the authentication, “`userAddress`” and “`wallet`”. If `userAddress` is present, only that value will be used for the authentication process, while if it is not there `wallet` will be used. However, in case both are present, there is no check which verifies that they are equal, meaning one address could be passed for `userAddress` and a different one for `wallet` - but only `userAddress` will be verified. This means that if an endpoint later reads the wallet value of the request, it will assume that address was authenticated as the user address, while in fact it could be any address.
Recommendation: Either remove the ambiguity in the check and only verify `userAddress`, or verify `userAddress` and `wallet` are equal when both are passed.

Resolution: It is now checked that both addresses are the same.

business/impl/bridge_estimator.py

BE1. Missing verification for price returned from API [medium] [resolved]

The bridge estimator calls the Stargate API to get the amount that will be received from the bridge and calculate the bridge fee from that. For slippage protection, a `dstAmountMin` value is passed to the API. However the result from the API is not being validated to ensure the API respected the `dstAmountMin` limit.
Running a test call (see below), it seems the API does not always respect the minimum amount, and even if it currently does, it should not be assumed as it could become a risk in the future.

The test call:

```
curl -X GET 'https://stargate.finance/api/v1/quotes?'\
'srcToken=0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE&'\
'srcChainKey=ethereum&'\ 'dstToken=0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE&'\
'dstChainKey=arbitrum&'\
'srcAddress=0x1234567890abcdef1234567890abcdef12345678&'\
'dstAddress=0xabcdef1234567890abcdef1234567890abcdef12&'\
'srcAmount=1000000000000000000&'\ 'dstAmountMin=950000000000000000000'
```

The result:


```
{"quotes":[{"route":"stargate/v2/taxi","error":null,"srcAmount":"10000000000000
00000","dstAmount":"998499000000000000","srcAmountMax":"27853729900000000000",
"dstAmountMin":"950000000000000000000","srcToken":"0xEeeeeEeeeEeEeeEeEeEeeEEEee
eeEeeeeeeEEeE","dstToken":"0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE","srcAdd
ress":"0x1234567890abcdef1234567890abcdef12345678","dstAddress":"0xabcdef123456
7890abcdef1234567890abcdef12","srcChainKey":"ethereum","dstChainKey":"arbitrum"
,"dstNativeAmount":"0","duration":{"estimated":180.828},"fees":[{"token":"0xEee
eeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE","chainKey":"ethereum","amount":"19327810
039623","type":"message"}],"steps":[{"type":"bridge","sender":"0x1234567890abcd
ef1234567890abcdef12345678","chainKey":"ethereum","transaction":{"data":"0xc7c7
f5b3000000000000000000000000000000000000000000000000000000000000008000000000000
0000000000000000000000000000000000000000011941b406f47000000000000000000000000000
0000000000000000000000000000000000000000000000000000000001234567890abcdef1
234567890abcdef12345678000000000000000000000000000000000000000000000000000000000
0000759e0000000000000000000000abcdef1234567890abcdef1234567890abcdef120000000
0000000000000000000000000000000000000000de0b6b3a764000000000000000000000000000
0000000000000000000000337fe5feaf2d1800000000000000000000000000000000000000000
000000000000000000000000e0000000000000000000000000000000000000000000000000000
0000000010000000000000000000000000000000000000000000000000000000000000120000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000","to":"0x77b2043768d28E9C9aB44E1aBfC95944bcE579
31","value":"1000019327810039623","from":"0x1234567890AbcdEF1234567890aBcdef123
45678"}}]},{"route":"stargate/v2/bus","error":null,"srcAmount":"100000000000000
0000","dstAmount":"998499000000000000","srcAmountMax":"278537299000000000000","
dstAmountMin":"950000000000000000","srcToken":"0xEeeeeEeeeEeEeeEeEeEeeEEEeeee
eEeeeeeeeEEeE","dstToken":"0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeeEEeE","srcAddr
ess":"0x1234567890abcdef1234567890abcdef12345678","dstAddress":"0xabcdef1234567
890abcdef1234567890abcdef12","srcChainKey":"ethereum","dstChainKey":"arbitrum",
"dstNativeAmount":"0","duration":{"estimated":480.828},"fees":[{"token":"0xEeee
eEeeeEeEeeEeeEEEeeeeEeeeeeeeeEEeE","chainKey":"ethereum","amount":"703168831
3867","type":"message"}],"steps":[{"type":"bridge","sender":"0x1234567890abcdef
1234567890abcdef12345678","chainKey":"ethereum","transaction":{"data":"0xc7c7f5
b3000000000000000000000000000000000000000000000000000000000000800000000000000
0000000000000000000000000000000000006653148bc0b000000000000000000000000000000
0000000000000000000000000000000000000000000000001234567890abcdef123
4567890abcdef12345678000000000000000000000000000000000000000000000000000000000
00759e0000000000000000000000abcdef1234567890abcdef1234567890abcdef12000000000
0000000000000000000000000000000000de0b6b3a764000000000000000000000000000000
0000000000000000000337fe5feaf2d1800000000000000000000000000000000000000000000
0000000000000000000e0000000000000000000000000000000000000000000000000000000000
0000001000000000000000000000000000000000000000000000000000000000000000012000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000010100000000000000000000000000000000000000000000000000
00000000000000","to":"0x77b2043768d28E9C9aB44E1aBfC95944bcE57931","value":"1000
007031688313867","from":"0x1234567890AbcdEF1234567890aBcdef12345678"}}]}]}
```


Recommendation: Verify the final amount from the Stargate API is not less than the minimum amount. Note that this also applies to `_fetch_bridge_routes` in `stargate.py` in contract interactions.

Resolution: The issue was resolved as recommended.

business/impl/planner.py

PL1. Wrong input value in swap action [medium] [resolved]

In `plan_deactivate_actions`, the `PendleSwapTool` is configured setting:

```
token_out=base_token_address,
```

The value for `amount` passed here is the value of `token_in` in base tokens - as the variable name suggests. This value comes from the code snippet here:

```
tools = await planner.plan_deactivate_actions(
      current_base_token=portfolio.get_base_token_balances_by_chain(),
      current_allocations=portfolio.get_current_allocations(),
```

```
            target_chain_id=target_chain,
        )
```

However, following the swap logic all the way to `pendle_approve_and_swap` in the `arma_chain` package, it will be passed as `amount_in` to the pendle router.

Recommendation: rename the param from `amount` to `amount_in` so the semantics are clear. And you should swap exactly all your input tokens, not some estimate of their value.
Resolution: The issue was resolved as recommended

business/impl/db/sql_writer.py

SQLW1. Missing activation_date on wallet update [low] [resolved]

The `update_agent_info` function does not update the activation date when the wallet class marks it as updated, which means its new value will not be saved when updated.
Recommendation: Save the `activation_date` of the wallet on `update_agent_info`.

SQLW2. Unused functions [info] [resolved]

The `activate_wallet`, `deactivate_wallet`, `update_wallet_status`, `update_last_reactivation_date`, `_convert_to_standard_datetime` functions are unused and do not integrate well with other parts of the system.
Recommendation: Remove unused code and avoid leaving code which is not properly implemented to avoid mistakenly calling it in the future.
Resolution: The issue was resolved as recommended. The functions were removed.

business/impl/db/sql_reader.py

SQLR1. Check current TVL can be manipulated [medium] [not resolved]

The `check_current_tvl` function relies on the deposit amounts saved from the users deposits. However, these deposit amounts are not being validated, and users can save any amount as their deposit amount. Thus these values should not be used as a trusted source for TVL calculation.
A second observation is that by checking only the Deposits, the `check_current_tvl` excludes the yield earned by users on their deposits, which should be counted in the TVL.

Recommendation: Validate the amounts of the users deposits before saving them. Also count the full managed value including yield towards the TVL.
Resolution: With the resolution of ABA1 and the disabling of the top-up endpoint, this issue is resolved.

SQLR2. Unused functions [info] [resolved]

The `_get_transactions_and_total`, `get_wallets`, `get_last_transaction_hashes`, `get_wallets_by_status`, `get_approved_dex_targets` functions are unused and unimplemented/ do not integrate well with other parts of the system.
Recommendation: Remove unused code and avoid leaving code which is not properly implemented to avoid mistakenly calling it in the future.
Resolution: The issue was resolved as recommended. The functions were removed.

business/impl/manager.py

M1. Wrong check for sufficient delta APR on run [low] [resolved]

The run function has a call to `_is_delta_apr_enough`, which is meant to verify the delta between the old and new APR is sufficiently big to execute a new plan, but the condition checking the delta APR is reversed.

Currently, the condition is:

```
if not opt_res or self._is_delta_apr_enough(opt_res):
```

When it should be:

```
if not opt_res or not self._is_delta_apr_enough(opt_res):
```

The reason is that the `_is_delta_apr_enough` will return True if the APR is *not* enough.
Recommendation: Fix the condition and the function as shown above.
Resolution: The issue was resolved as recommended.

M2. Funds may get stuck on a low yield protocol due to reallocation constraint [low] [not resolved]

The `run` function optimizes the funds allocation checks using the `_is_reallocation_allowed` function. The check passes if the current portfolio value is higher than historical max plus a threshold, presumably to make sure a protocol first covered any migration costs which were involved in switching to it before considering moving again to another protocol. However, this means that in case of a sudden and unexpected drop in market yield in the active protocol, this reallocation constraint will prevent switching from the protocol, potentially indefinitely, risking leaving the funds on a protocol which doesn't generate much or any yield.
Recommendation: Consider limiting the time `_is_reallocation_allowed` can block new plan execution.
Resolution: The issue was acknowledge but not addressed

business/impl/constants.py

C1. Missing mainnet values [medium] [resolved]

The entire `constants.py` file seems to be missing the values for mainnet, which is intended to be supported, meaning mainnet can not be fully supported by the system as is.
Recommendation: Add mainnet values as needed.
Resolution: The issue was resolved as recommended.

business/impl/optimizer.py

O1. Market names are treated as unique while not necessarily being unique [high][resolved]

In multiple instances, the market `name` is used as a unique identifier for a market. However, these names are not necessarily unique. For example, the first 2 markets returned by the API (https://api-v2.pendle.finance/core/v1/1/markets/active) are both named "`wstETH`" - and that is just on main net.
This issue has ramifications in different parts of the code, some of which play an essential role. Issue O2 is a direct result from this issue. Another example is in `_extract_optimization_results`, where different Markets that have the same `name` will be assigned the same `allocation_value` in the `allocations` mapping, will be counted double when calculating `weightedAPY_final` or when calculating the `cash_position`, etc.

Recommendation: Use a unique identifier for markets instead of the market's `name` - for example, a combination of its address and the id of the chain it lives on.
Resolution: The issue was resolved as recommended

O2. Error in setting up constraints for market_apys [high] [resolved*]

In `_prepare_optimization_data`, the function loops over all markets `market` and all chains `chain_id`, calculates the APY for allocating funds to `market`. This function takes into account the costs of bridging any funds that are not allocated on `market.chain` (which is the chain that the market lives on) from `chain_id` to `market.chain`.
It saves the result under a key `(market.name, chain_id)`

```
        for market in markets:
            for chain_id in available_chains:
                funding_key = (market.name, chain_id)
                try:
                    xs, ys, raw_apy = await
self._build_piecewise_for_funding_source(
                        market,
                        chain_id,
                        T_total,
                        current_allocations,
                    )
                    breakpoints[funding_key] = (xs, ys)
                    final_market_apys[funding_key] = raw_apy
```

This choice of saving the results will skip a lot of relevant information: the results for each combination of `(market.name, market.chain, chain_id)` are calculated, but the results are saved under a key that does not take the value for `market.chain` into account.
For example, suppose the system supports two chains, `1` and `8453`, and two markets: a `wstETH` market on mainnet (call it `market1`) and a `wstETH` market on Base (call it `market2`). Suppose the function is called with the values in the given order (i.e. `markets = [market1, market2]` and `available_chains` is equal to `[1, 8453]`. The function will now save the breakpoints under two different funding keys:

`breakpoints[("wstETH", 1)]`: the APYs of allocation to `market2`, taking into account the cost of bridging missing funds from main net

`breakpoints[("wstETH", 8453)]`: the APYs of allocation to `market2` without taking into account any bridging costs

No breakpoints are saved for `market1`.

Recommendation: If you want to keep this logic (i.e. calculate APY estimates for all `source_chain` and market pair, you should include the unique identifier for the market in the `funding_key` (and not just its `name`).
Resolution: The code was not changed, but the team stated that in the first release, they will only deploy to Arbitrum

O3. _get_effective_apy_for_funding does not take into account that funds may be allocated from other chains [medium] [resolved*]

The function `_get_effective_apy_for_funding(market, source_chain, amount_in, current_allocations)` plays a central role in building the constraints of the optimization problem. It returns the APY of investing `amount_in` tokens in the market, and subtracts the cost of bridging tokens that are currently not allocated to that `market` from the given `source_chain`. These values are then used as inputs to the optimization problem.

However, the effective apy for investing `amount_in` for any two `market` and `source_chain` pairs are not independent. To see this, suppose the agent has 1 ETH on Arbitrum, and 1 ETH on Base, that each yield 15%. There is a new market on Main to which no funds are currently allocated that offers an APY for investing 1 ETH of 20%, which are the best rates available. However, investing 2 ETH in this market on Main yields only 10%. In the current implementation of the optimization problem, the effectiveAPY for funding 1 ETH from Arbitrum is 20%, and so is the effective APY for funding 1 ETH from Base. The algorithm will optimize for both of these values, and therefore allocate 2 ETH to the market on Main net, for a net yield of 10% (rather than the 17.5% that an optimal allocation would yield)
Recommendation: Add constraints to the optimization problem that take the APY of the combined investment into account.
Resolution: The team has acknowledged the issue. It is a limit case that may result in a suboptimal resolution, and will be addressed in a future release. In the first launch, only Arbitrum will be available, so the issue does not occur

O4. Bridging costs are not correctly treated in _get_effective_apy_for_funding [low] [resolved]

The function `_get_effective_apy_for_funding(market, source_chain, amount_in, current_allocations)` plays a central role in building the constraints of the optimization problem. It returns the APY of investing `amount_in` tokens in the market, and subtracts the cost of bridging tokens that are currently not allocated to that `market` from the given `source_chain`. This is done by calculating the bridge_cost as a percentage and then "amortizing" the costs relative to the expiry date:

```
annual_cost_impact = cost_percentage * (365 / days_to_expiry)
```

The result is then subtracted from the base_apy

```
return base_apy - bridge_impact
```

1. A first problem with this treatment is that allocations are typically *not* held until expiry, so this will systematically underestimate the costs of bridging relative to the APY (and so overestimate the APY).
2. Another problem is the fact that in the current code, the bridge costs are treated as if they were spread out over the year, while in reality they are applied to the principal at the moment of bridging. This is especially relevant if bridge costs are high. For a somewhat artificial example, suppose the bridge costs are 50% and the APY is 50% as well, and the allocation is held for a year. The current calculation gives a net APY of 0%, while in reality, the net return is a loss of 25%.

Recommendation: For the first problem, instead of calculating bridge cost impact relative to the expiry of the market, use an estimate of the length of time an allocation typically will be held. For the second problem. For the second issue, use a pattern such as this:

```
apr_per_day = (1 + apy) ** (1 / 365) - 1
principal = amount_in * bridge_cost_absolute
days = estimate_how_long_to_hold_this_allocation_given_market_expiry()
return principal * (1+ apr_per_day) ** days
```

Resolution: The estimated or predicting “holding time” of an investment is set to 60 days - which the team states is a typical period. The team has not addressed the second problem we noted, but states that the `_get_effective_apy_for_funding` function is only used to *compare* investment returns rather than calculate an absolute value, so the current approximation is satisfactory.

O5. Failing to apply a constraint is silently ignored [low] [resolved]

The `_setup_allocation_constraints` will try to apply any added `extra_constraints`, and in case it fails to apply a constraint, it simply skips it. While `extra_constraints` are not currently updated at any point, this could be a risk in the future, as constraints can be critical and should not be ignored.
Recommendation: Raise an error instead of skipping the constraint if you fail to apply it.
Resolution: The issue was resolved as recommended

O6. Wrong value calculated in initial weighted APY [low] [resolved]

The `market_apy` is used to calculate the APYs on the current position with current allocation of each market.

```
      market_apy = max(final_market_apys.get((market.name, chain_id), 0.0) for
chain_id in available_chains)
```

But `final_market_apys` itself is calculated on the basis of the full `T_total` instead of just the current allocation, meaning the result of `weightedAPY_initial` will be incorrect.
Recommendation: Replace `final_market_apys` with APYs based on the current allocations.
Resolution: The issue was resolved as recommended

O7. MAX_FEE may underestimate costs [low] [resolved]

The `MAX_FEE` is defined in `costants.py` as

```
      MAX_FEE = 100_000_000 * USDC_USDT_DECIMALS
```

and so is equal to 1e12 WEI. For ETH, this is not a lot, and will almost surely underestimate the actual costs.
Recommendation: Use a much higher `MAX_FEE`, or improve error handling instead of utilizing `MAX_FEE` to handle error cases.
Resolution: MAX_FEE is now set to MAX_FEE = 0.003 * ETH_DECIMALS.
Note that the use of MAX_FEE is ambiguous - it is returned as an upper limit when estimated bridge costs, but alos when it is impossible to calculate the costs. Upstream, it is then sometimes interpreted as an error condition ((e.g. in the optimizer, a value of MAX_FEE is used as signal to not bridge any funds to that market), and sometimes treated as a “good value” (e.g. in _extract_optimization_results, it is used to calculate the value of real_bridge_costs.

O8. Piece calculation may be generated even when getting APY fails [low] [not resolved]

The `_build_piecewise_for_funding_source` function checks if the `APY_0` and `APY_T` difference is less than `tolerance`, and will run the full piecewise calculation if it isn’t.
The `_get_effective_apy_for_funding` function which is used to calculate the APY values may return either 0 or -1 when the calculation fails (the return value depends on the error). If calculation of both APYs fails for the same reason, their value will be the same, the difference between them will be 0. This means that with tolerance of 0 (which is always used currently),

having both APY calculations fail means their difference is equal to tolerance, and will continue to the full piecewise calculation.
Recommendation: Properly handle errors in APY calculation instead of returning 0 or -1.
Resolution: The issue was not resolved.

O9. Swap data result should use effectiveAPY instead of impliedAPY [low] [resolved]

The function `_get_effective_apy_for_funding` queries the pendle API for information about the APY when swapping base tokens into the desired PT tokens. The Pendle API returns two pertinent values: `impliedAPY` and `effectiveAPY`. Currently, the first is used.

We believe this is not correct. The value for `impliedAPY` represents the APY when holding the token - it is directly based on the current market price of the token, and does not depend on the parameters of the trade. The value for `effectiveAPY` takes into account the given slippage, fees, and the amount you're swapping.

In other words, as it currently stands, the costs for swapping (such as slippage and fixed costs) are not taken into account when calculating the APY, which will result in a systemic underestimation of the actual costs for switching.
Recommendation: Use the `effectiveAPY` instead of `impliedAPY`
Resolution: The issue was resolved as recommended.

job/pendle_job/main.py

JM1. Using of non-existent DEFAULT_THRESHOLD [low] [resolved]

The `threshold` value in the `create_wallet_groups` function is set to the value of `settings.THRESHOLD`, and if that is `None` (as it is now), it will use `settings.DEFAULT_THRESHOLD`, but that value is not even defined, which will raise an error.
Recommendation: Define `DEFAULT_THRESHOLD` in settings.
Resolution: The relevant lines were removed.

pendle/business/impl/tools/*

T1. Post execution of tools may save empty tx hash [low] [resolved]

The `post_execution` function of `tools` extracts the tx hash from a response object, then calls `update_transaction` to save the transaction data. However, if getting the tx hash was not possible, it uses an empty string as the tx hash, which is wrong.
Recommendation: Avoid saving tx data if tx hash was not found in response.
Resolution: The issue was resolved as recommended.

T2. Code duplication between tools [info] [partially resolved]

Most of the code of each of the tools is nearly identical, and so it will be better to move common code to the `Tool` class to avoid duplication.
Recommendation: Move the common code to the `Tool` class.
Resolution: The issue was partially resolved. Some of the common code has been moved as recommended.

Arma Contract Interactions

packages/arma_chain/src/arma_chain/chain_ops/enums.py

CE1. Missing BRIDGE action [info] [resolved]

The `TxAction` enum is missing the `BRIDGE` action used in the Pendle repo.
Recommendation: Add the `BRIDGE` action.
Resolution: The issue was resolved as recommended.

packages/arma_chain/src/arma_chain/chain_ops/tx_manager.py

TXM1. Unnecessary approve call on transfer [low] [not resolved]

The `approve_and_transfer` function first calls approve, then transfers the funds to the destination address.
However, approval is not needed when a normal transfer is made, and this call can be removed.

Recommendation: Remove the `approve_and_transfer` function and use just the `transfer` method.
Resolution: The issue was not resolved.

packages/cross_chain/src/cross_chain/pendle/pendle.py

CPP1. Slippage is not being validated [high] [resolved]

The `get_swap_transaction` function calls `_validate_slippage` to limit the allowed slippage up to `MAX_SLIPPAGE`. However, the call does not save the result into the `slippage` variable as it should, meaning the slippage check does not take effect.
Recommendation: Save the result of `_validate_slippage` into `slippage`.
Resolution: The issue was resolved as recommended.