F Y E O

Theoriq

Repo: https://github.com/Moonsong-Labs/tq-oracle

Security Review Update: January 22, 2026

Reviewer: balthasar@gofyeo.com

Security Code Review TQ Oracle

New security issues, 10

After the development team implemented the latest updates, FYEO conducted a review of the modifications. The primary goal of this evaluation was to ensure the continued robustness of the program's security features, safeguarding user funds and maintaining the overall integrity of the program.

General Updates:

The update expanded the TQ Oracle system's asset discovery capabilities by adding support for multiple new DeFi protocols and implementing an adapter chaining architecture. The system now supports Uniswap V3 and V4 liquidity positions, Pendle PT and LP tokens with oracle-based conversion to underlying assets, and multiple Aave V3 instances simultaneously (enabling tracking of both Aave and Spark pools). The adapter chaining mechanism allows sequential transformation of wrapped tokens, where one adapter discovers a position (like an Aave aToken) and subsequent adapters in the chain convert it to underlying assets (like Pendle converting PT tokens or ERC4626 converting vault shares). This enables the system to handle complex nested positions such as Aave-supplied Pendle PT tokens or Aave-supplied ERC4626 vault tokens.

The update also introduced specialized handling for Real World Asset (RWA) vault tokens with different liquidity profiles and exit strategies. A custom sNUSD adapter was added to track both idle vault token balances and assets in redemption queues (with 10-day cooldown periods), ensuring accurate TVL reporting for tokens with constrained exit paths. The system now distinguishes between three patterns: high-liquidity tokens priced via market rates (using CoinGecko), instant-redeem vault tokens converted via ERC4626 standard, and queue-based redemption tokens requiring custom adapters to track multiple asset states. The configuration system was enhanced to support per-subvault adapter chains, multi-instance protocol support, and flexible pricing strategies across CoinGecko, Chainlink, CoW Swap, and Pyth validation sources.

Chainlink Adapter Missing Validation Checks

Finding ID: FYEO-TQO-01
Severity: **Medium**
Status: **Remediated**

Description

The Chainlink price adapter fetches ETH/USD prices using `latestRoundData()` but only validates that `answer > 0`. It does not perform validation checks recommended by Chainlink's documentation. Missing validations include: checking if `updatedAt != 0` (indicating valid data), verifying `answeredInRound >= roundId` (detecting stale rounds), implementing a staleness threshold (ensuring price freshness), and validating phase consistency (detecting oracle upgrades). Without these checks, the adapter could use stale, invalid, or manipulated price data, leading to incorrect TVL calculations and potentially exploitable conditions.

According to Chainlink's security documentation, all consumers of `latestRoundData()` should validate the returned data to ensure it's fresh and valid.

Proof of Issue

File name: src/tq_oracle/adapters/price_adapters/chainlink.py

```
async def _get_eth_usd_price(self) -> tuple[int, int]:
    """Get ETH/USD price from Chainlink oracle."""
    ...

    _, answer, _, updated_at, _ = round_data  # Unpacks but doesn't validate
most fields

    if answer <= 0:  # Only validation performed
        raise ValueError(f"Invalid Chainlink price: {answer}")

    logger.debug(
        "Chainlink ETH/USD: price=%d, decimals=%d, updated_at=%d",
        answer,
        decimals,
        updated_at,  # Logged but not validated
    )

    return answer, decimals
```

Severity and Impact Summary

Without staleness checks, the adapter could use prices that are hours or days old. If `updatedAt == 0`, it indicates the round has no valid data, but the adapter would still use the price. If `answeredInRound < roundId`, it means the round hasn't been fully answered yet, and the price may be from a previous round.

Recommendation

Ensure the round has valid data. Verify the round is complete and not stale. Reject prices older than a configurable threshold. Compare `updatedAt` against the current block timestamp to detect staleness.

Inconsistent Exception Handling, Silent Partial Failures in Idle Balances Adapter

Finding ID: FYEO-TQO-02
Severity: **Low**
Status: **Remediated**

Description

The idle balances adapter uses `return_exceptions=True` in `asyncio.gather()` when fetching individual asset balances and then silently logs and skips exceptions without raising. This goes against the system's fail-fast principle where accurate TVL reporting requires complete data. If fetching the balance for any supported asset fails (RPC timeout, contract error), the adapter returns partial results without indicating that data is incomplete. This can lead to understated TVL values in reports.

The system has an exception handling policy at the pipeline level where `_process_adapter_results()` raises `ValueError` if any adapter fails. However, the idle balances adapter undermines this by catching exceptions internally and returning partial success, making the pipeline think everything succeeded when data is missing.

Proof of Issue

File name: src/tq_oracle/adapters/asset_adapters/idle_balances.py

```
asset_results = await asyncio.gather(*asset_tasks, return_exceptions=True)

assets: list[AssetData] = []
for asset_addr, result in zip(supported_assets, asset_results):
    if isinstance(result, Exception):
        logger.error(  # Only logs, doesn't raise
            "Failed to fetch balance for asset %s in subvault %s: %s",
            asset_addr,
            subvault_address,
            result,
        )
    elif isinstance(result, AssetData):
        assets.append(result)

logger.debug("Fetched %d L1 asset balances for subvault", len(assets))
return assets  # Returns partial results without indicating failure
```

Severity and Impact Summary

If fetching the balance for any asset fails, the TVL report will be understated without any indication of incompleteness.

Recommendation

Apply a consistent pattern*, modify `fetch_assets()` to collect failed assets and raise `ValueError` if any asset fetch fails, matching the pattern used in `fetch_all_assets()`.

Missing Configuration Validation

Finding ID: FYEO-TQO-03
Severity: **Low**
Status: **Remediated**

Description

The configuration system lacks comprehensive validation at both the settings level and adapter initialization level. Multiple validation gaps exist: no duplicate detection for vault tokens, market addresses, or token addresses across adapter configurations; no validation of address format consistency; no bounds checking on numeric parameters; no validation that required fields are mutually consistent; and no detection of conflicting configurations across multiple adapter instances. These gaps can lead to double-counting of assets (inflated TVL), silent configuration errors, and runtime failures that could have been caught at startup.

The system uses Pydantic for basic type validation but does not implement custom validators to enforce business logic constraints. Adapters accept configuration dictionaries without validating uniqueness, completeness, or logical consistency of the values.

Proof of Issue

File name: src/tq_oracle/adapters/asset_adapters/rwa/erc4626_vault.py

```
# Vaults configuration
if "vaults" in overrides:
    self.vaults = overrides["vaults"]
elif adapter_config.vaults:
    self.vaults = adapter_config.vaults
else:
    self.vaults = {}  # No validation of vault_token uniqueness
```

File name: src/tq_oracle/adapters/asset_adapters/pendle.py

```
# Markets configuration
if "markets" in overrides:
    self.markets = overrides["markets"]
elif adapter_config.markets:
    self.markets = adapter_config.markets
else:
    self.markets = {}  # No validation of market address uniqueness
```

File name: src/tq_oracle/adapters/asset_adapters/aave_v3.py

```
# Supply tokens (aTokens)
if "supply_tokens" in overrides:
    self.supply_tokens = overrides["supply_tokens"]  # No duplicate check
    logger.debug("Using override supply_tokens: %s", self.supply_tokens)
elif adapter_config.supply_tokens:
    self.supply_tokens = adapter_config.supply_tokens  # No duplicate check
    logger.debug("Using config supply_tokens: %s", self.supply_tokens)
else:
    self.supply_tokens = AAVE_V3_SUPPLY_TOKENS_MAINNET

# Borrow tokens (variable debt)
if "borrow_tokens" in overrides:
    self.borrow_tokens = overrides["borrow_tokens"]  # No duplicate check
    logger.debug("Using override borrow_tokens: %s", self.borrow_tokens)
elif adapter_config.borrow_tokens:
    self.borrow_tokens = adapter_config.borrow_tokens  # No duplicate check
    logger.debug("Using config borrow_tokens: %s", self.borrow_tokens)
```

File name: src/tq_oracle/adapters/asset_adapters/rwa/erc4626_vault.py

```
async def _try_convert_vault_token(self, asset: AssetData) -> AssetData | None:
    """Try to convert asset if it's an ERC4626 vault token we know about."""
    vault_address = asset.asset_address.lower()  # Assumes lowercase works

    # Check if this is a vault token in our config
    for vault_name, vault_config in self.vaults.items():
        if vault_config["vault_token"].lower() == vault_address:  #
Case-insensitive comparison
```

File name: src/tq_oracle/settings.py

```
class StakewiseAdapterSettings(BaseModel):
    """Configuration options for StakeWise adapter defaults."""
    stakewise_exit_queue_start_block: int = 0  # No validation that it's
reasonable
    stakewise_exit_max_lookback_blocks: int =
STAKEWISE_EXIT_MAX_LOOKBACK_BLOCKS  # No max limit
```

File name: src/tq_oracle/adapters/asset_adapters/rwa/erc4626_vault.py

```
for vault_name, vault_config in self.vaults.items():
    if "vault_token" not in vault_config:  # Runtime check, not config
validation
        logger.warning(
            "Skipping ERC4626 vault %s: missing 'vault_token' config",
            vault_name,
        )
        continue
```

Severity and Impact Summary

Duplicate vault/market/token configurations cause the same balance to be fetched and counted multiple times, directly inflating TVL reports. Invalid configurations (malformed addresses, missing fields, out-of-bounds values) are not detected until runtime, potentially causing failures during operations. Configuration errors manifest as runtime failures or incorrect calculations, making root cause analysis difficult. Invalid configurations could cause adapter failures, incomplete data collection, or system crashes during production runs.

Recommendation

Implement comprehensive validation for all config values.

Systemic Address Normalization Inconsistency Across Codebase

Finding ID: FYEO-TQO-04
Severity: **Low**
Status: **Remediated**

Description

A systemic inconsistency exists throughout the codebase regarding Ethereum address normalization. Different modules use different strategies (lowercase, checksum, or raw/as-is), leading to multiple concerns: cache misses for the same address in different formats, failed address comparisons, incorrect asset aggregation, and redundant RPC calls. The problem affects adapters, utilities, reports, and processors. Some code uses `addr.lower()` for normalization, some uses `Web3.to_checksum_address()`, and some stores addresses without normalization, creating a fragmented approach that potentially causes subtle bugs and performance degradation.

Proof of Issue

File name: src/tq_oracle/adapters/price_adapters/coingecko.py

```
async def get_token_decimals(self, token_address: str) -> int:
    if token_address in self._decimals_cache:  # Case-sensitive lookup
        return self._decimals_cache[token_address]

    self._decimals_cache[token_address] = decimals  # Stores as-is
```

File name: src/tq_oracle/adapters/price_adapters/manual.py

```
async def get_token_decimals(self, token_address: str) -> int:
    if token_address in self._decimals_cache:  # Case-sensitive
        return self._decimals_cache[token_address]

    self._decimals_cache[token_address] = decimals  # Stores as-is
```

File name: src/tq_oracle/report/subvault_breakdown.py

```
_DECIMALS_CACHE: dict[str, int] = {}  # Global cache

async def get_token_decimals(token_address: str, config: OracleSettings) ->
int:
    if token_address in _DECIMALS_CACHE:  # Case-sensitive
        return _DECIMALS_CACHE[token_address]

    _DECIMALS_CACHE[token_address] = decimals  # Stores as-is
```

File name: src/tq_oracle/adapters/price_adapters/coingecko.py

```
# Uses lowercase for token_ids
for addr, cg_id in COINGECKO_DEFAULT_IDS.items():
    self.token_ids[addr.lower()] = cg_id  # Lowercase

# But cache uses raw addresses
self._decimals_cache[token_address] = decimals  # Not normalized
```

File name: src/tq_oracle/adapters/price_adapters/manual.py

```
self.manual_prices = {
    addr.lower(): price for addr, price in config.manual_prices.items()
}  # Lowercase for prices, but cache uses raw addresses
```

File name: src/tq_oracle/report/subvault_breakdown.py

```
# Groups by lowercase
asset_groups[asset.asset_address.lower()].append(asset.amount)

# Compares with lowercase
if isinstance(addr, str) and addr.lower() == asset_addr:

# But price lookup uses linear search with lowercase
for key in prices.prices.keys():
    if key.lower() == asset_addr:
```

File name: src/tq_oracle/utils/asset_conversion.py

```
if (
    usd_asset_address
    and underlying_address.lower() != usd_asset_address.lower()   # Lowercase
):
```

File name: src/tq_oracle/adapters/asset_adapters/pendle.py

```
pt_address = asset.asset_address.lower()   # Lowercase

if market_pt_address.lower() == pt_address:   # Lowercase comparison
```

Severity and Impact Summary

Same token address in different formats causes cache misses across multiple adapters, resulting in redundant RPC calls for decimals. Same asset in different formats may not aggregate correctly, leading to incorrect TVL calculations. Address comparisons may fail when comparing addresses in different formats, causing logic errors.

Recommendation

Choose one normalization approach for the entire codebase. All addresses should be normalized immediately when entering the system (adapter initialization, function parameters, API responses). Normalize keys in all cache dictionaries.

Fragile Block Timestamp Caching in sNUSD Adapter

Finding ID: FYEO-TQO-05
Severity: **Informational**
Status: **Remediated**

Description

The sNUSD adapter caches block timestamps in a simple dictionary without keying by block number. While this works in the current single-block execution model, the cache persists across multiple `fetch_assets()` calls. If the adapter instance is reused with different `block_identifier` values, the cache will return stale timestamps from previous blocks, causing incorrect cooldown state calculations.

Proof of Issue

File name: src/tq_oracle/adapters/asset_adapters/rwa/snusd.py
Line number: 147

```
async def _get_block_timestamp(self) -> int:
    """Get timestamp of current block."""
    if self._block_timestamp_cache is not None:
        return self._block_timestamp_cache

    block = await self._rpc(self.w3.eth.get_block, self.block_number)
    timestamp = int(block["timestamp"])
    self._block_timestamp_cache = timestamp
    return timestamp
```

Severity and Impact Summary

Incorrect cooldown state calculations if block number changes between calls. Wrong classification of pending vs claimable amounts.

Recommendation

Apply pattern consistently. Review adapters for similar caching patterns and apply consistent future-proof logic.

Inconsistent Semaphore Configuration

Finding ID: FYEO-TQO-06
Severity: **Informational**
Status: **Remediated**

Description

Multiple adapters use inconsistent configuration attribute names for semaphore initialization, with some using `getattr(config, "max_calls", 5)` while others use `config.rpc_max_concurrent_calls`.

Proof of Issue

File name: src/tq_oracle/adapters/asset_adapters/stakewise.py
Line number: 126

```
self._rpc_sem = asyncio.Semaphore(getattr(config, "max_calls", 5))
```

File name: src/tq_oracle/adapters/asset_adapters/streth.py
Line number: 69

```
self._rpc_sem = asyncio.Semaphore(getattr(self.config, "max_calls", 5))
```

Severity and Impact Summary

User’s `rpc_max_concurrent_calls` setting is ignored in affected adapters. Inconsistent behavior across adapters.

Recommendation

Standardize semaphore configuration.

Missing HTTP Session Reuse in CoinGecko Adapter

Finding ID: FYEO-TQO-07
Severity: **Informational**
Status: **Remediated**

Description

The CoinGecko adapter uses `requests.get()` directly without a `requests.Session` object, causing a new TCP/TLS connection to be established for every API request. This results in unnecessary TLS handshake overhead, increased latency, and wasted resources. Using a session would enable connection pooling and TLS session reuse, improving performance for repeated API calls.
In the Asset Conversion Utility, the Web3 Provider is recreated per call.

Proof of Issue

File name: src/tq_oracle/adapters/price_adapters/coingecko.py
Line number: 200

```
response = await asyncio.to_thread(
    requests.get,
    url,
    params=params,
    headers=headers,
    timeout=10.0,
)
```

File name: src/tq_oracle/utils/asset_conversion.py

```
async def convert_shares_to_assets_erc4626(
    ...
) -> tuple[str, int]:
    """Convert ERC-4626 vault shares to underlying asset amount."""
    w3 = Web3(Web3.HTTPProvider(config.vault_rpc_required))  # New connection
every call

    ...
```

Severity and Impact Summary

Increased latency per API request.

Recommendation

Create a session in `__init__` as an instance variable during adapter initialization. Replace `requests.get()` with `self._session.get()` in `_fetch_prices_batch()`.

Missing Retry Logic for Uniswap V4 GraphQL

Finding ID: FYEO-TQO-08
Severity: **Informational**
Status: **Remediated**

Description

The Uniswap V4 adapter makes GraphQL requests to The Graph subgraph to fetch position data but does not implement any retry or backoff logic for these requests. Unlike RPC calls in other adapters which use `@backoff.on_exception` decorators, the GraphQL requests use raw `requests.post()` wrapped in `asyncio.to_thread()` without retry protection. This means any transient network error, subgraph timeout, rate limiting, or temporary service unavailability will cause the entire adapter to fail immediately, even though these errors are typically recoverable with retry logic.

The Graph subgraphs can experience temporary issues including rate limiting, node synchronization delays, query timeouts, and network connectivity problems. Without retry logic, the adapter is more fragile and could fail to collect Uniswap V4 position data during temporary outages, leading to incomplete TVL reports.

Proof of Issue

File name: src/tq_oracle/adapters/asset_adapters/uniswap/uniswap_v4.py

```
resp = await asyncio.to_thread(
    requests.post,
    self._graph_url(),
    json=payload,
    timeout=20,
)  # no retry or backoff logic
resp.raise_for_status()  # raises immediately on any HTTP error
data = resp.json()
if "errors" in data:
    raise ValueError(f"Subgraph query errors: {data['errors']}")  # no retry
```

Severity and Impact Summary

Any temporary network issue, subgraph timeout, or rate limiting causes immediate adapter failure without retry.

Recommendation

Wrap the GraphQL request function with `@backoff.on_exception` decorator similar to other HTTP requests in the codebase.

Missing Timeout Protection on RPC Calls

Finding ID: FYEO-TQO-09
Severity: **Informational**
Status: **Remediated**

Description

Asynchronous RPC calls using `asyncio.to_thread()` lack timeout protection. This pattern appears in nearly all adapters, utilities, and reports.

If an RPC endpoint becomes slow, unresponsive, or malicious, these calls hang, blocking the pipeline and preventing TVL calculation.

Furthermore, across all adapters where the `_rpc()` method executes a sleep delay executes within the semaphore's context manager. This means the semaphore slot remains occupied during the sleep period, preventing other tasks from acquiring it. This design conflates two separate concerns (concurrency limiting via semaphore and rate limiting via delay) and results in reduced effective concurrency.

The semaphore's purpose is to limit concurrent operations, while the delay's purpose is to space out requests over time. By sleeping while holding the semaphore, the code effectively reduces the configured concurrency limit.

Proof of Issue

```
async def _rpc(self, fn, *args, **kwargs):
    """Execute RPC call with throttling and retry."""
    async with self._rpc_sem:
        try:# no timeout
            return await asyncio.to_thread(fn, *args, **kwargs)
        finally:
            delay = self._rpc_delay + random.random() * self._rpc_jitter
            if delay > 0:
                await asyncio.sleep(delay)  # holding semaphore during sleep
```

StakeWise Adapter, Aave V3 Adapter, sNUSD, Pendle, ERC4626, strETH, CoinGecko, …

Severity and Impact Summary

The service will hang when RPC providers do not respond, which means TVL reports fail to publish.

Recommendation

Consider an appropriate response to request timeouts. Add alternative routes or other fail overs.

Unbounded Multicall Requests

Finding ID: FYEO-TQO-10
Severity: **Informational**
Status: **Acknowledged**

Description

The strETH adapter constructs a single multicall with all subvault addresses without chunking. If the number of subvaults is large, the single `aggregate(calls)` operation may exceed RPC node limits, causing the entire request to fail. This creates a single point of failure.

Proof of Issue

File name: src/tq_oracle/adapters/asset_adapters/streth.py
Line number: 105

```
async def _fetch_assets(self, subvault_addresses: list[str]) ->
list[AssetData]:
    # ...
    calls = []
    for subvault in subvault_addresses:
        calls.append([
            Web3.to_checksum_address(collector.address),
            collector.encode_abi(
                "getDistributions",
                args=[...],
            ),
        ])

    call_results = (
        await self._rpc(
            self.multicall.functions.aggregate(calls).call,
            block_identifier=self.block_number,
        )
    )[1]
```

File name: src/tq_oracle/adapters/asset_adapters/streth.py
Line number: 169-189

```
async def fetch_all_assets(self) -> list[AssetData]:
    # ...
```

Severity and Impact Summary

Complete adapter failure if subvault count exceeds node limits.

Recommendation

Verify this implementation works for the expected use case. Split calls into batches if required.

Commit Hash Reference:

For transparency and reference, the security review was conducted on the specific commit hash for the Theoriq repository. The commit hash for the reviewed versions is as follows:

a63d765dfd09089632ce1cc4b45f14eefcc72d50

Remediations were submitted with the commit hash 2a6f4a34ba590a6a15c3654dee32289d978c2f32.

Conclusion:

In conclusion, the security aspects of the Theoriq program remain robust and unaffected by the recent updates. Users can confidently interact with the protocol, assured that their funds are well-protected. The commitment to security exhibited by the development team is commendable, and we appreciate the ongoing efforts to prioritize the safeguarding of user assets.