GA GUARDIAN

Bounce

Automation Layer & Contract Updates

Security Assessment

December 10th, 2025

Summary

Audit Firm Guardian

Prepared By Felipe Gomez, Balazs, Arif Khan

Client Firm Bounce

Final Report Date December 10, 2025

Audit Summary

Bounce engaged Guardian to review the security of their Automation Layer & Contract Updates. From the 27th of October to the 13th of November, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Code is clean, well-structured, and adheres to best practices. Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

Note: Fixes to the findings uncovered in the remediation review have not been reviewed by Guardian.

Security Recommendation: Guardian recommends a follow-up security review of the protocol at a finalized, frozen commit. While confidence in the smart contract layer is high, a closer review of the automation layer would offer helpful additional assurance.

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only 1 Significant issue was uncovered per week. Design patterns are sound, and test coverage is strong. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity issues. Varied Low severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1-2 High/Critical findings per engagement week.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities. Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week. Or additional High/Critical findings uncovered in remediation review which have not been resolved and confirmed by Guardian.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Bounce
Language	Solidity
Codebase	https://github.com/bounce-tech/bounce-automation, https://github.com/bounce-tech/bounce-contracts
Commit(s)	Automation Layer Main Review commit: e93dee5ba0c5d59ef97e0f274611bb11b3597fa7 Bounce Contract Main Review Commit: 82cdc08a1e29571083f55f9572de1c83cde5f1c0 Automation Layer Remediation Review commit: b187210bf1b501229cf3374a69e3afaaf8edc94d Bounce Contract Remediation Review Commit: 648c37fb830d6a760810ea7f1c92f85bf037ee4d

Audit Summary

Delivery Date	December 10, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	1	0	0	0	0	1
Medium	11	0	0	2	0	9
Low	23	0	0	3	1	19
Info	18	0	0	6	0	12

Audit Scope & Methodology

Automation Layer: https://github.com/bounce-tech/bounce-automation

Contract Updates:
https://github.com/bounce-tech/bounce-contracts/compare/4016d3a5fb136ba0012a4f6fcdd5748af0e873a1...82cdc08a1e29571083f55f9572de1c83cde5f1c0

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Incorrect Position Distribution Amount	Logical Error	High	Resolved
M-01	Adding Agent Wallets Will Silently Revert	Logical Error	Medium	Resolved
M-02	Retroactive Streaming Fee Charge	Logical Error	Medium	Acknowledged
M-03	GKE Control Plane Exposed	Access Control	Medium	Resolved
M-04	Spot Assets Swaps Slippage Too High	Configuration	Medium	Resolved
M-05	Unsafe Deserialization	Logical Error	Medium	Resolved
M-06	Temporary Redemption Execution Loop	Logical Error	Medium	Resolved
M-07	Sandwich Attack On Large Position Updates	MEV	Medium	Acknowledged
M-08	Minimum Margin Ratio	Configuration	Medium	Resolved
M-09	No Delay Between Bridge Funds Steps	Unexpected Behavior	Medium	Resolved
M-10	Partial TVL Calculation In Case Of Failures	Logical Error	Medium	Resolved
L-01	LT Redeployment Could Lead To Unprocessed Events	Unexpected Behavior	Low	Resolved
L-02	Unencrypted Redis Connection	Configuration	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-03	Unauthenticated Redis Connection	Access Control	Low	Resolved
L-04	Use Of Mutable Tag For Base Image	Best Practices	Low	Resolved
L-05	Insufficient RPC URL Validation	Validation	Low	Resolved
L-06	No Slippage Protection	MEV	Low	Acknowledged
L-07	Maximum Token Leverage Not Enforced	Logical Error	Low	Resolved
L-08	No Validation Across Config Files	Validation	Low	Resolved
L-09	Application Fails To Start Due To Config Errors	Configuration	Low	Resolved
L-10	Service Related Settings Not Updated On Change	Unexpected Behavior	Low	Resolved
I-01	Errors Not Used	Superfluous Code	Info	Resolved
I-02	Temporary DoS In Leveraged Token Actions	DoS	Info	Acknowledged
I-03	Only Last Locker Registered	Configuration	Info	Resolved
I-04	Add Account Script Logs Private Key To Console	Best Practices	Info	Resolved
I-05	Potential Private Key Exposure In Shell Script	Best Practices	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-06	Incorrect UsdcSpotToAsset Class Name	Informational	Info	Resolved
I-07	Duplicated RPC URL Validation Logic	Best Practices	Info	Resolved
I-08	No Fallback API For Price Updates	Suggestion	Info	Acknowledged
I-09	Batching Events Could Save Gas	Informational	Info	Acknowledged
I-10	Margin Event Uses Wrong Objective	Superfluous Code	Info	Acknowledged
I-11	Regex Fails On Alphanumeric Tokens	Logical Error	Info	Resolved

H-01 | Incorrect Position Distribution Amount

Category	Severity	Location	Status
Logical Error	High	fund_location.py: 59-65	Resolved

Description

The fund distribution contains balances in evm, spot, perps and position locations. This distribution is later used in different areas of the code, specially to calculate leveraged token TVL and target leverage.

However, the `hypercore-position` distribution uses position size instead of the margin used.

Consequently, following impact arise:

• Inflated TVL, as it will combine USDT/USDC balances in evm, spot and perps with the perp asset position size in USDC terms, causing wrong comparison checks with the minimum TVL to stop exposure.

• Incorrect target leverage calculation, as the `position_margin_usdc` will be assigned to the notional value of the position instead of the margin used, leading to a totally unexpected value.

• Objective type will not be calculated correctly, as the `hypercore-position` fund is inflated, so will always try to send funds back to EVM to align with the smart contract buffer configurations.

Recommendation

The `hypercore-position` distribution should use the position margin instead of position notional.

Resolution

Bounce Team: The issue was resolved in PR#files.

M-01 | Adding Agent Wallets Will Silently Revert

Category	Severity	Location	Status
Logical Error	Medium	LeveragedToken.sol: 266-267	Resolved

Description

When adding an API wallet on the `LeveragedToken` contract, the agent's name is obtained by concatenating the token's symbol, leverage, side, "AGENT" word and agent's slot.

The initial token deployments will use BTC, ETH, HYPE and SOL. The agent's names for these tokens will have between 13-15 characters, depending on the token and leverage (i.e. `ETH5S_AGENT_2` or `HYPE10L_AGENT_1`).

There are tokens with longer symbols, like `FARTCOIN`. With a 2 digit leverage, the agent's name will be longer: `FARTCOIN10L_AGENT_1`. However, agent's names can't exceed 16 characters, so any call to `setAgent` for certain tokens succeed in EVM but revert in Core.

Recommendation

Consider reducing the agent's character length to support all tokens.

Resolution

Bounce Team: The issue was resolved in PR#141.

M-02 | Retroactive Streaming Fee Charge

Category	Severity	Location	Status
Logical Error	Medium	GlobalStorage.sol: 170	Acknowledged

Description PoC

Owner can change streaming fee at any time using `GlobalStorage`. In case the streaming fee is increased, the subsequent `checkpoint` call invoked by any action computes `timeElapsed` since the old `lastCheckpoint` and charges fees for the entire period during which the fee was lower, effectively retroactively charging fees at an increased rate.

Recommendation

Make sure to call checkpoint in live leveraged tokens before increasing streaming fees.

Resolution

Bounce Team: Acknowledged.

M-03 | GKE Control Plane Exposed

Category	Severity	Location	Status
Access Control	Medium	main.tf: 37-45	Resolved

Description

The Google Kubernetes Engine (GKE) cluster is configured to expose its control plane (the Kubernetes API server) to the entire public internet. The master_authorized_networks_config is set to allow access from 0.0.0.0/0, which is a wildcard for all IP addresses.

Risk:

Exposing the Kubernetes control plane to the public internet is a critical security risk and significantly increases the cluster's attack surface.
1. Increased Attack Surface: It makes the Kubernetes API server a direct target for attackers anywhere in the world.
2. Vulnerability to Attacks: The exposed control plane is vulnerable to a wide range of attacks, including brute-force password attempts, credential stuffing, and exploits targeting any potential vulnerabilities
(including zero-days) in the Kubernetes API server.
3. Total Cluster Compromise: The control plane is the "brain" of the Kubernetes cluster. An attacker who successfully compromises the control plane can gain complete administrative control over the entire cluster. This would allow them to deploy malicious workloads, exfiltrate or destroy sensitive data, and use the cluster's resources for their own purposes.

Recommendation

Access to the GKE control plane should be restricted to the smallest possible set of trusted IP addresses.

1. Enable and Configure Master Authorized Networks: The most immediate fix is to replace 0.0.0.0/0 with a specific, limited set of IP address ranges. These should be the public IPs of trusted networks from which
administrators or automation will access the cluster (e.g., corporate office networks, CI/CD runner IPs, or bastion host IPs).

• Example:

```
1       master_authorized_networks_config {
2         cidr_blocks {
3           cidr_block   = "203.0.113.5/32"
4           display_name = "Corporate Office"
5         }
6       }
```

1. Use a Private Cluster (Best Practice): For a much higher level of security, configure the GKE cluster as a Private Cluster. This provisions the control plane with an internal IP address, making it inaccessible from the public internet. Access to the control plane would then be managed through a bastion host, a VPN, or IAP (Identity-Aware Proxy) tunneling within your Google Cloud VPC.

Resolution

Bounce Team: The issue was resolved in PR#66.

M-04 | Spot Assets Swaps Slippage Too High

Category	Severity	Location	Status
Configuration	Medium	config-mainnet.json: 10	Resolved

Description

Both `UsdcSpotToAsset` and `SpotAssetToUsdc` classes are initialized with a slippage value fetched from the env configs.

However, the mainnet json config file sets 2% for the `spot_conversion_slippage`. Therefore, any spot swaps (USDC <=> baseAsset) will allow 2% slippage, which is too high for stablecoin swaps.

Recommendation

Consider lowering the slippage for spot market orders.

Resolution

Bounce Team: The issue was resolved in PR#61.

M-05 | Unsafe Deserialization

Category	Severity	Location	Status
Logical Error	Medium	src/models/leveraged_token_key.py: 32	Resolved

Description

The `from_config` classmethod is responsible for deserializing a leveraged token from either a dictionary or a string. It contains two weaknesses that can lead to application errors or the processing of invalid data.

Unsafe Dictionary Deserialization: When parsing a dictionary, the code accesses keys directly (e.g., `value['leverage']`). If a provided dictionary is missing any of the required keys (`perp_asset`, `leverage`, `is_long`), the application will raise an unhandled `KeyError` and crash.

Overly Permissive String Parsing: The regular expression used to parse string inputs (`^([A-Za-z]+)...`) allows any sequence of one or more letters for the asset name. This is too broad and does not enforce a standard ticker format (e.g., 2-6 uppercase letters).

It could allow the creation of invalid keys like "`MyInvalidAsset10L`", which may cause silent failures or errors in downstream systems that expect a standard asset ticker.

Recommendation

The deserialization logic should be hardened to be more resilient to malformed input.

For Dictionary Input: Use the `.get()` method to safely access keys. Check if any required values are missing and raise a single, clear `ValueError` that lists the missing fields.

For String Input: Tighten the parsing to ensure that the leverage is not extracted from given string through regex,

Resolution

Bounce Team: The issue was resolved in PR#91.

M-06 | Temporary Redemption Execution Loop

Category	Severity	Location	Status
Logical Error	Medium	blockchain.py: 292	Resolved

Description

The redemption service is in charge of checking pending redemptions and executing them in order of appearance.

However, there could be cases where a `PREPARE_REDEEM` event is processed, but while the `REMOVE_MARGIN` event flow is executed, the objective steps either fail or they are skipped due to specified rules.

Therefore, a pending redemption will be created with insufficient evm contract funds to execute the redemption. The following scenario might take place:

- User prepares redemption
- Perp limit order fails due to slippage, next steps are skipped do to min transfer amounts (rules).
- Executor sees pending redemption, tries to execute it
- As there is not enough balance in contract, it skips user and the for loop ends
- As tx does not revert, automation thinks redemption was processed
- One iteration every 60 sec, which fetches pending redemptions again, and process repeats.
- Executor will keep trying to execute redemptions, spending HYPE in each tx.
- Loop will continue until a new margin change event occurs, to recalculate distribution and steps.

Keep in mind this scenario might also appear if a user prepares redeem when there is no active position (i.e. funds were bridged to EVM but limit order failed), as the rebalance service will not trigger a margin change event if position is empty.

Recommendation

Either add a retry mechanism so funds are bridged back to EVM, or avoid calling `executeRedemptions` more than once with same params, until a new margin changing event occurs

Resolution

Bounce Team: The issue was resolved in PR#48.

M-07 | Sandwich Attack On Large Position Updates

Category	Severity	Location	Status
MEV	Medium	hyperliquid_api.py: 362	Acknowledged

Description

When executing a perps market order, scripts will calculate limit price and slippage, but no validation is done for the notional value.

In case there is a large mint or redeem, position increase/decrease value might be large, causing a significant price jump in the order book.

Although there is a limit price calculation and slippage protection, attackers can back run the EVM transaction and front run the hyperliquid order, and profit from the price jump.

Recommendation

Consider using TWAP orders if order notional value is above certain threshold.

Resolution

Bounce Team: Acknowledged.

M-08 | Minimum Margin Ratio

Category	Severity	Location	Status
Configuration	Medium	minimum_margin_position_rule.py: 35	Resolved

Description

The current environment configuration uses `minimum_margin_ratio=0.15` and `position_to_open`.

This suggests that the scripts will try to open a 9x leverage initial position, but then validates that the margin used is above 15% of the projected notional.

However, a 9x leverage position yields to a 11,11% margin ratio, which will always be higher than the minimum configured margin ratio, disallowing position opening due to `MinimumMarginPositionRule`

Recommendation

Consider reducing the `minimum_margin_ratio` to a value below 11,11% to allow positions to be opened.

Resolution

Bounce Team: The issue was resolved in PR#47.

M-09 | No Delay Between Bridge Funds Steps

Category	Severity	Location	Status
Unexpected Behavior	Medium	contract_to_spot.py: 39	Resolved

Description

During `PositionService` execution of multi-step fund transfers (e.g., `ContractToSpot`), there is no delay or block confirmation wait after an EVM bridge transaction.

As a result it could happen, that the next step could fail because the balance change is not immediately reflected there, resulting in skipped steps, stuck funds.

Recommendation

Add some delay after the EVM to Core bridge tx executed to make sure all previous tx are processed and finalized.

Resolution

Bounce Team: The issue was resolved in PR#68.

M-10 | Partial TVL Calculation In Case Of Failures

Category	Severity	Location	Status
Logical Error	Medium	`src/services/position_service.py` (400-410, `_get_current_distribution` method)	Resolved

Description

The application's Total Value Locked (TVL) calculation process, specifically within the `_get_current_distribution` method, is designed to be resilient to individual data source failures.

When attempting to fetch fund balances from various FundLocations (e.g., blockchain contracts, Hyperliquid exchange accounts), any exception that occurs during an API or RPC call for a specific location is caught.

Instead of halting the entire TVL calculation, the system logs a warning and proceeds to the next `FundLocation`, effectively excluding the funds from the failed source.

• `src/services/position_service.py` (400-410, `_get_current_distribution` method)

Recommendation

1. Evaluate Acceptable Risk: Determine if a partial TVL calculation is an acceptable operational state. For critical financial decisions, it might be preferable for the entire TVL calculation to fail if a significant data source is unavailable.

2. Enhanced Alerting: If a partial TVL is deemed acceptable, implement more prominent logging and alerting mechanisms (e.g., `PagerDuty`, Slack notification) when a data source fails to fetch balances. This ensures operators are immediately aware of the incomplete data.

3. Graceful Degradation Strategy: If a data source is critical, consider a strategy where the system either:
- Uses the last known good balance for a short period.
- Temporarily pauses operations that rely on the TVL until all critical data sources are restored.

4. Clearer Reporting: Ensure that any UI or reporting of the TVL clearly indicates if it is based on a partial dataset due to data source failures.

Resolution

Bounce Team: The issue was resolved in PR#92.

L-01 | LT Redeployment Could Lead To Unprocessed Events

Category	Severity	Location	Status
Unexpected Behavior	Low	blockchain_executeRedemptions.py: 75	Resolved

Description

The automation layer always queries the latest list of the leverage tokens by calling the factory state.

In case an LT gets redeployed (only condition to not have any `marginUsed` for it) the given address gets removed from the factory lts list.

This could cause issues in case there is a pending redeem or other kind of events in the automation side for the old lt token instance.

Recommendation

In the factory's `redeployLt` function you could check if there are any pending redemption in the given lt instance, or somehow on automation side cache and gracefully process all the pending events for an lt token before start processing the new ones.

Resolution

Bounce Team: The issue was resolved in PR#70.

L-02 | Unencrypted Redis Connection

Category	Severity	Location	Status
Configuration	Low	redis_storage.py: 16-27	Resolved

Description

The application communicates with the Redis server over a plain TCP connection, which is unencrypted. All data exchanged between the application and Redis is transmitted in plaintext.

Recommendation

1. Configure the Redis server to support and require SSL/TLS connections.

2. Update the application's connection logic in `src/storage/redis_storage.py` to include the necessary SSL parameters (e.g., `ssl=True`, `ssl_ca_certs`) to establish a secure, encrypted connection.

Resolution

Bounce Team: The issue was resolved in PR#58.

L-03 | Unauthenticated Redis Connection

Category	Severity	Location	Status
Access Control	Low	storage_type.py: 40-50	Resolved

Description

The application connects to the Redis server without providing a password or any other form of authentication.

Risk:

This allows any user or process with network access to the Redis server to gain full control over the data it holds. An unauthorized actor could read, modify, or delete data, disrupt application logic (like leader
election), or execute administrative commands on the Redis server.

Evidence:
1. Explicit Log Message: The code contains a log message that explicitly states the connection is unauthenticated.
• File: src/storage/storage_type.py
• Lines: 35-36
• Code Snippet:

```
1       // ...
2       34    logger.info(
3       35        f'Multi-process mode: Connecting to Redis without authentication at {redis_host}:{redis_port_int}/{redis_db_int}')
4       36
5       37    storage = RedisStorage(
6       // ...
```

1. Redis Client Instantiation: The redis.Redis client is instantiated without the password parameter, confirming that no credentials are being used.
• File: src/storage/redis_storage.py
• Lines: 18 and 21 (same as the unencrypted finding)
• Code Snippet:

```
1        // ...
2        15        redis_kwargs = {
3        16            'host': host,
4        17            'port': port,
5        18            'db': db,
6        19            'socket_connect_timeout': socket_connect_timeout,
7        20            'socket_timeout': None
8        21        }
9        22        self._redis_blocking = redis.Redis(**redis_kwargs) # No 'password' parameter
```

```
10        // ...
11        25         self._redis = redis.Redis(**redis_kwargs) # No 'password' parameter
12        // ...
```

Recommendation

1. Enable Redis Authentication: Configure the Redis server to require a password by setting the requirepass directive.
2. Use Secrets Management: Store the Redis password securely in your secrets management system.
3. Update Application Code: Modify the application to retrieve the Redis password and provide it in the redis.Redis constructor call using the password parameter.

Resolution

Bounce Team: The issue was resolved in PR#58.

L-04 | Use Of Mutable Tag For Base Image

Category	Severity	Location	Status
Best Practices	Low	Dockerfile: 1	Resolved

Description

The Dockerfile at deployments/Dockerfile uses python:3.11-slim as the base image for the container. This is a "floating" tag, meaning that the underlying image it points to can be updated over time by the image
Maintainers.

Risk:

Using mutable tags like latest or 3.11-slim introduces several risks:

1. Unpredictable Builds: Your CI/CD pipeline might pull a newer version of the base image without your knowledge. This new version could contain breaking changes that cause your application to fail at build time or,
worse, at runtime.

2. Vulnerability Management: It becomes difficult to track and manage vulnerabilities. A new version of the base image could introduce new security vulnerabilities (CVEs). Because your build is not pinned to a specific
version, you can't be certain which version of the base image a particular build of your application is using, making vulnerability remediation challenging.

3. Inconsistent Deployments: Different environments (e.g., development, staging, production) could end up running on different underlying base images, even if they are all built from the same Dockerfile. This violates
the principle of immutable infrastructure and can lead to "it works on my machine" problems.

Recommendation

Pin your base images to a specific, immutable version using a digest (@sha256:...). This guarantees that your builds are always reproducible and that you are using a known, vetted version of the base image.

1. Find the Digest: First, pull the image you want to use and find its digest:

```
1    docker pull python:3.11-slim
2    docker inspect python:3.11-slim | grep "Digest"
```

This will give you a sha256 digest.
1. Update the Dockerfile: Use the full image name, tag, and digest in your FROM instruction.
• Example:

```
1       FROM python:3.11-slim@sha256:a14e84333778030615168a46595f40e4d33e3519d18a7070a87972618543485c
2
3       WORKDIR /app
4       ...
```

By pinning to the digest, you ensure that every build of your application will use the exact same base image, making your builds more secure and reliable.

Resolution

Bounce Team: The issue was resolved in PR#81.

L-05 | Insufficient RPC URL Validation

Category	Severity	Location	Status
Validation	Low	src/api/blockchain.py:444, src/scripts/script_init.py:144, src/framework/base_framework.py:107 src/framework/base_framework.py:117	Resolved

Description

The validation logic for RPC URLs is critically insufficient. It only checks if a URL string begins with the https:// prefix and fails to perform any validation on the URL's hostname. This allows URLs pointing to internal or restricted network locations to be accepted and used by the application.

This flaw creates a risk for Server-Side Request Forgery (SSRF) vulnerability. An attacker who can control the value of the RPC URL configuration (e.g., via an environment variable or a compromised secret store) can force the application to send requests to arbitrary network endpoints from the server's perspective.

This can be exploited to:

• Scan Internal Networks: Discover live hosts and open ports on the internal network that are not exposed to the internet.
• Access Internal Services: Interact with internal databases, admin panels, or other services that may not require authentication because they are assumed to
be unreachable externally.
• Steal Cloud Credentials: Make requests to the cloud provider's metadata service (e.g., 169.254.169.254) to exfiltrate temporary credentials, which can then be used to compromise other cloud resources.

Recommendation

The RPC URL validation logic must be enhanced to perform strict hostname validation.
1. Implement Strict Host Validation: The validation function should parse the URL to extract its hostname.
2. Resolve and Verify IP Address: The hostname should be resolved to its IP address. The function must then verify that the IP address is a public,
non-reserved address.
3. Reject Restricted IPs: The validation must explicitly reject any URL whose hostname resolves to:
• Loopback addresses (127.0.0.0/8)
• Private IP ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
• Link-local addresses (169.254.0.0/16)
4. Apply Centrally: This robust validation should be implemented in a single, centralized function (as recommended in the "Duplicated Logic" finding) and used by all parts of the application that process RPC URLs.

Resolution

Bounce Team: The issue was resolved in PR#93.

L-06 | No Slippage Protection

Category	Severity	Location	Status
MEV	Low	LeveragedToken.sol: 200	Acknowledged

Description

The `prepareRedeem` → `executeRedemptions` redemption path allows users to request redemptions asynchronously via executors, but provides no slippage protection for the user during execution.

• In `prepareRedeem`, the user locks their LT tokens and receives credit based on the current exchange rate.
• Later, in `executeRedemptions`, the executor processes the redemption using the exchange rate at execution time, which may have moved adversely (especially for large redemptions or volatile markets).
• There is no `minBaseAmount` parameter in `prepareRedeem`, and no check in `executeRedemptions` to ensure the user receives at least a minimum acceptable base asset amount.
• A whale redeeming a large position could suffer significant slippage if market conditions deteriorate between `prepareRedeem` and execution.

This violates the principle of user-controlled slippage tolerance and could cause loss of funds for the user.

Recommendation

Add a `minBaseAmount` parameter to `prepareRedeem`
• Store this value per user in storage.

In `executeRedemptions`, check slippage for the user
Allow users to update their slippage or cancel redemption
Document that large redemptions should use direct `redeem()` for tight slippage control, or accept delayed execution risk.

Resolution

Bounce Team: Acknowledged.

L-07 | Maximum Token Leverage Not Enforced

Category	Severity	Location	Status
Logical Error	Low	config-mainnet-position-service-main.json	Resolved

Description

The `Factory` is in charge of deploying new leveraged tokens. It uses `_validateLeverage` to check if the target token leverage is within 1x and max leverage supported in Hyperliquid for given perp asset.

The automation layer will not support tokens above 9x leverage, due to the `position_to_open` config value.

Furthermore, as the target leverage will normally be above the token specified leverage, if a token is deployed with a leverage close to 9x, it won't be supported as well, due to the `smart_contract_buffer` of 20% (i.e. 8x / 0.8 = 10x > 9x).

If any unsupported token leverage is deployed and added to the automation script management, there is no validation on the token leverage. If a mint event occurs on this leveraged token, funds will be bridged to core, but position will never be opened due to leverage rules.

Recommendation

If `smart_contract_buffer=0.2` and `position_to_open=9`, the max token leverage supported is 7.2x. Consider adding this check to the factory smart contract directly.

Alternatively, consider validating the token leverage in the automation scripts, when reading `tokens_supported` from the config files.

Resolution

Bounce Team: The issue was resolved in PR#73.

L-08 | No Validation Across Config Files

Category	Severity	Location	Status
Validation	Low	config-mainnet-position-service-main.json: 2	Resolved

Description

The system uses multiple JSON config files (e.g., for different environments, services, or instances) that define `TOKENS_SUPPORTED` lists.

However, there is no enforcement that these lists match across files/services, and no validation prevents duplicate tokens within a single list.

Could lead to double event processing.
Token mismatch, missing event processing.

Recommendation

Create a single, centralized `supported_tokens.json` file containing all the `TOKENS_SUPPORTED` list. + Add Duplicate Check when reading the list of supported tokens.

Resolution

Bounce Team: The issue was resolved in PR#75.

L-09 | Application Fails To Start Due To Config Errors

Category	Severity	Location	Status
Configuration	Low	#src/framework/base_framework.py, line 101, src/services/position_service.py, line 137	Resolved

Description

The application fails to launch in the single-process local development mode (python src/main.py). The root cause is a series of configuration loading and validation issues stemming from the fact that the architecture is primarily designed for a multi-process production environment.

The system incorrectly attempts to find and validate service-specific configuration files instead of gracefully sharing the main environment config (config-testnet.json), leading to three distinct, sequential crashes on startup.

Issue #1: Incorrect Config File Path
Symptom: The application crashes immediately with ValueError: Config file config-testnet-main-main.json could not be found.
Root Cause: The BaseFramework._create_instance_config_manager method incorrectly constructs a file path intended for a multi-process environment. It combines the network, the service's config name (e.g., position-service), and the instance name (main), resulting in a path to a file that does not exist in a local setup. It should instead load the main config-testnet.json file.

Issue #2: Strict Schema Validation Failure
Symptom: After fixing the file path, the application crashes during service initialization with ValueError: Mandatory field mark_prices_supported is missing from config.
Root Cause: Although all services now correctly load config-testnet.json, each service validates this shared file against its own unique schema which expects service-specific fields (e.g., mark_prices_supported for PriceCacheService). Since these fields are not in the generic environment config, the mandatory field check in ConfigSnapshot.__post_init__ fails.

Issue #3: Unhandled None Values in Service Logic
Symptom: After relaxing the schema validation, the application crashes in a service's validation callback (PositionService.on_config_validation) with TypeError: 'NoneType' object is not iterable.
Root Cause: By making the schema fields optional for local runs, the config manager now correctly returns None for missing fields. However, the downstream code within the services does not account for this None value and attempts to iterate over it directly, causing a TypeError.

Recommendation

Correct Config Loading for Local Mode: In src/framework/base_framework.py, the _create_instance_config_manager was modified to detect the single-process mode (instance_name == 'main'). In this mode, it now:

Correctly loads the main environment config file (e.g., config-testnet.json).

Dynamically creates a new ConfigSchema with an empty mandatory_fields set. This allows the configuration to be loaded without failing validation on missing service-specific fields.

Improve Service Robustness: In src/services/position_service.py, the on_config_validation method was updated to handle None values returned from the config. It now provides default empty iterables (e.g., ... or [], ... or {}) to prevent TypeError when a configuration field is not present.

Improve Error Messaging: The error message in _create_config_manager was improved to clearly state which file it was looking for and where it searched, making future debugging easier.

These changes ensure the application can adapt its configuration strategy for the local development environment, leading to a successful startup.

Resolution

Bounce Team: The issue was resolved in PR#76.

L-10 | Service Related Settings Not Updated On Change

Category	Severity	Location	Status
Unexpected Behavior	Low	base_framework.py: 47	Resolved

Description

Each service (e.g., `PositionService`, `RebalanceCheckService`) has its own `configs.json` file but currently only the `config-{network}.json` being watched by the config manager.

For example if a supported token gets removed for a service, the change won't be detected by the config manager.

Recommendation

Start the config manager for each service on startup.

Resolution

Bounce Team: The issue was resolved in PR#76.

I-01 | Errors Not Used

Category	Severity	Location	Status
Superfluous Code	Info	ILeveragedToken.sol	Resolved

Description

The following errors in `ILeveragedToken` are declared but never used:

• `InsufficientCredit`

• `LastAgent`

• `AgentAlreadyUsed`

Recommendation

Remove unused errors

Resolution

Bounce Team: The issue was resolved in PR#142.

I-02 | Temporary DoS In Leveraged Token Actions

Category	Severity	Location	Status
DoS	Info	LeveragedToken.sol: 161	Acknowledged

Description

Previously, a protocol deadlock could appear if a whale redeemed 100% of the base assets in the contract, as there will be no funds to pay streaming fee in `_checkpoint`.

However, users will still face a temporary DoS, as leaving contract without any base assets is still possible. Any transaction involving `mint` or `prepareRedeem` will still revert until the agent wallets bridge in funds from `HyperCore`.

Recommendation

As there is no trivial solution that does not involve refactoring a big part of the code, consider documenting this to users, so they are aware of these temporal DoS.

Resolution

Bounce Team: Acknowledged.

I-03 | Only Last Locker Registered

Category	Severity	Location	Status
Configuration	Info	Tokenomics.s.sol: 50-66	Resolved

Description

The deployment script deploys three distinct `Locker` contracts (A, B, C) but calls `globalStorage.setLocker()` for each, overwriting the prior address. As a result, only the last (Locker C) remains discoverable via `GlobalStorage`, while Locker A and B are not registered.

Any on-chain or off-chain components that rely on `GlobalStorage` to reference all lockers will miss A and B, potentially breaking claims/operations for those allocations and orphaning distributed tokens.

Recommendation

Provide dedicated setters in `GlobalStorage` for multiple lockers (e.g., `setLockerA/B/C` or a setter that stores an array). Update the script to call the correct distinct setters. If a single locker is intended, deploy only one and distribute accordingly.

Resolution

Bounce Team: The issue was resolved in PR#143.

I-04 | Add Account Script Logs Private Key To Console

Category	Severity	Location	Status
Best Practices	Info	add-account.sh: 123	Resolved

Description

The generated accounts private key being logged as plain text to the console, which could be retrieved if the system gets compromised.

Recommendation

Remove the logging part of the pk.

Resolution

Bounce Team: The issue was resolved in PR#71.

I-05 | Potential Private Key Exposure In Shell Script

Category	Severity	Location	Status
Best Practices	Info	upload-account-to-secrets.sh: 6-10	Resolved

Description

The shell script located at `scripts/internal/upload-account-to-secrets.sh` is designed to upload a private key to Google Secret Manager. It accepts the private key directly as a command-line argument.

Risk:

Passing secrets as command-line arguments is a highly insecure practice. It can lead to the private key being exposed in several ways:

1. Shell History: Most shells are configured to save a history of executed commands. The command, including the private key, will be stored in plaintext in the user's shell history file (e.g., `.bash_history`).

2. Process List: While the script is running, the full command, including the private key, can be viewed by any user on the system who can list running processes (e.g., by using the ps aux command).

3. Logs: If the command is executed as part of an automated script or CI/CD pipeline, it is likely to be logged, again exposing the private key in plaintext.

An attacker who gains access to the user's account, the system's process list, or the relevant logs could easily retrieve the private key.

Recommendation

Modify the script to avoid passing the private key as a command-line argument. Instead, prompt the user to enter the private key interactively or read it from a file.

Example (Interactive Prompt):

```
1 # ...
2 read -sp "Enter private key: " PRIVATE_KEY
3 echo
4 # ...
5 # Add secret version with private key
6 echo -n "${PRIVATE_KEY}" | gcloud secrets versions add "${SECRET_ID}" --data-file=- …
```

Using read -s prevents the typed key from being displayed on the screen and from being stored in the shell history. This significantly reduces the risk of accidental exposure.

Resolution

Bounce Team: The issue was resolved in PR#77.

I-06 | Incorrect UsdcSpotToAsset Class Name

Category	Severity	Location	Status
Informational	Info	usdc_spot_to_asset.py: 24	Resolved

Description

The `UsdcSpotToAsset` class appears to be a direct copy of `SpotAssetToUsdc` — its `name()` method still returns the old class name.

Recommendation

Name function should return `UsdcSpotToAsset` instead.

Resolution

Bounce Team: The issue was resolved in PR#65.

I-07 | Duplicated RPC URL Validation Logic

Category	Severity	Location	Status
Best Practices	Info	src/api/blockchain.py: 444, src/scripts/script_init.py: 144 src/framework/base_framework.py: 107 src/framework/base_framework.py: 117	Resolved

Description

The logic responsible for parsing a comma-separated string of RPC URLs and performing basic validation is duplicated in at least three separate locations within the codebase. This indicates a lack of a centralized utility for handling this common data-processing task.

The primary impact of this issue is on code quality, maintainability, and operational risk:

• Increased Maintenance Overhead: When the validation logic needs to be updated—whether for a bug fix or a feature enhancement—developers must find and modify every instance of the duplicated code. This is inefficient and error-prone.

• Risk of Inconsistent Behavior: It is easy for a developer to update the logic in one location but miss the others. This can lead to different parts of the application behaving inconsistently, making debugging difficult.

• Compounded Security Risk: If a security flaw exists in the duplicated logic (as is the case here), the risk is compounded because a patch must be applied correctly to all locations. Missing even one location leaves the system vulnerable.

Recommendation

1. Centralize the Logic: Create a single, well-defined function (e.g., `parse_and_validate_rpc_urls`) in a shared location, such as `src/common/utils.py`.

2. Refactor for Reuse: Refactor the code in the identified locations to import and use this new, centralized function, ensuring that all RPC URL processing is handled consistently across the application.

Resolution

Bounce Team: The issue was resolved in PR#64.

I-08 | No Fallback API For Price Updates

Category	Severity	Location	Status
Suggestion	Info	hyperliquid_subscriber.py	Acknowledged

Description

The `HyperLiquidSubscriber` relies exclusively on the Hyperliquid `WebSocket` API (Info client) for real-time price updates (mark price, mid price, etc.). While it includes stale detection and automatic reconnection, there is no fallback data source if:

• The Hyperliquid `WebSocket` is down for an extended period

• The API endpoint is rate-limited or blocked

• Network partitions prevent reconnection

This could result in stale price data. Even with reconnection logic, price updates cease entirely during outages — there is no secondary oracle or REST polling fallback

Recommendation

• Add a fallback REST polling mechanism using Hyperliquid's HTTP API

• Activate fallback when stale threshold exceeded AND reconnection fails > N times

Or optionally use different system oracles not hyperliquid related

Resolution

Bounce Team: Acknowledged.

I-09 | Batching Events Could Save Gas

Category	Severity	Location	Status
Informational	Info	position_service.py	Acknowledged

Description

The PositionService processes mint and redeem events independently via EventProcessor with per-token queues (queue_identifier based on LeveragedTokenKey). This means:

• A mint of 1000 USDC → triggers full fund transfer + position adjustment

• A redeem of 1000 USDC shortly after → triggers another full transfer in the opposite direction

Even though the net effect is zero, both events are fully processed, resulting in:

• Unnecessary gas fees (EVM contract calls)

• Unnecessary Hyperliquid orders (spot/perp trades)

• Increased slippage risk

There is no mechanism to:

• Batch opposing events (mint + redeem)

• Cancel or net out pending events before execution

• Skip redundant processing when net TVL change is below threshold

This is especially costly in high-frequency mint/redeem scenarios (e.g. arbitrage, rebalancing).

Recommendation

Add event batching & netting in EventProcessor
Batch events per block or time window

Resolution

Bounce Team: Acknowledged.

I-10 | Margin Event Uses Wrong Objective

Category	Severity	Location	Status
Superfluous Code	Info	objective_type.py: 129-131	Acknowledged

Description PoC

Currently, `ADD/REMOVE_MARGIN` events for existing positions emit a margin event with `is_leverage_change = False`.

This results in using the `KEEP_SMART_CONTRACT_FUND_WITHIN_BUFFER` `objectiveType`, which triggers full `AllocationPlanner` runs and causes unnecessary API calls, calculations, and processing.

Rebalancing may still be needed in some cases, but not always — only when the added or removed margin causes the position to move outside the buffer. In those cases, the timed rebalance checker can handle the adjustment separately.

Recommendation

Use `ADJUST_POSITION_ONLY` when position already exists and `is_leverage_change=false`.

Resolution

Bounce Team: Acknowledged.

I-11 | Regex Fails On Alphanumeric Tokens

Category	Severity	Location	Status
Logical Error	Info	leveraged_token_key.py: 32	Resolved

Description

The leveraged token key should have a specific format to guarantee a match here:

`match = re.match(r'^([A-Za-z]+)([1-9]\d*)([LlSs])\$', value)`.

Although this pattern will match currently supported tokens, it won't handle some edge cases, specially with alphanumeric tokens:

• if first character of the token name is a number, regex will not find a match and reverts (i.e. `0G` `4`)

• if the token has a number at the end of the name (i.e. `SPX6900`) and we want to short it 3x, which would be `SPX69003S`, the leverage value is extracted from the 2nd match group (matching numbers) in the regex pattern, which will lead to a 69003x leverage value.

Recommendation

Document this issue in the code, explaining why certain tokens are not supported.

Resolution

Bounce Team: The issue was resolved in PR#72.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
M-01	Inability To Remove Activated Agent Wallets	DoS	Medium	Resolved
L-01	Global Storage Owner Risk	Best Practices	Low	Acknowledged
L-02	Missing Token Support Validation	Validation	Low	Partially Resolved
L-03	Incorrect Redemption Execution Check	Unexpected Behavior	Low	Resolved
L-04	No Core Settlement Verification	Logical Error	Low	Resolved
L-05	Incomplete Private Key Cleanup Mechanism	Censoring	Low	Resolved
L-06	Potential Exposure Of Private Key Through Logs	Best Practices	Low	Resolved
L-07	Potential Private Key Exposure Via Shell Memory	Best Practices	Low	Resolved
L-08	Insecure File Creation Window (Race Condition)	Unexpected Behavior	Low	Resolved
L-09	"Fail Open" On DNS Failure	Error	Low	Resolved
L-10	Potential DNS Rebinding Vulnerability (TOCTOU)	Best Practices	Low	Acknowledged
L-11	Thread-Unsafe Global State Mutation	Best Practices	Low	Resolved
L-12	IPv6 Validation Bypass	Logical Error	Low	Resolved

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
L-13	Flawed Deduplication Logic In Telegram Alerting	Logical Error	Low	Resolved
I-01	Unnecessary Gas Spent For Executions	Gas Optimization	Info	Resolved
I-02	Unsupported Leveraged Tokens	Informational	Info	Acknowledged
I-03	Deployment Script Private Key Handling	Best Practices	Info	Resolved
I-04	Insecure Private Key Handling	Best Practices	Info	Resolved
I-05	Gaming The Order Of The Redemption Queue	Logical Error	Info	Acknowledged
I-06	Potential Info Disclosure Through Telegram API	Best Practices	Info	Resolved
I-07	Missing API Rate Limiting In Telegram Alerting	Best Practices	Info	Resolved

M-01 | Inability To Remove Activated Agent Wallets

Category	Severity	Location	Status
DoS	Medium	LeveragedToken.sol: 292	Resolved

Description

The protocol owner is able to add agent wallets to the `LeveragedTokens` using `setAgent`. This function now verifies if the contract address is already activated and if the agent wallet is NOT activated, to prevent silent reverts in Hyperliquid Core transaction.

Although the agent wallet activation check was added as an additional precaution, it adds a new DoS attack, as this agent can't be removed if the agent is activated in Core.

Therefore, if the agent wallet is compromised or the wallet needs to be replaced, anyone can just activate it and trigger a revert in the `setAgent` EVM transaction.

Recommendation

Remove the `HyperCore` activation check for the agent wallet, and instead perform this check off chain before adding new agents.

Resolution

Bounce Team: The issue was resolved in PR#152.

L-01 | Global Storage Owner Risk

Category	Severity	Location	Status
Best Practices	Low	GlobalStorage.sol: 51	Acknowledged

Description

The `GlobalStorage` owner is set to `msg.sender` during deployment. Besides setting protocol config params, this address will also be allowed to perform critical changes to the bounce token, leveraged tokens, among others.

However, the `msg.sender` will be the foundry scripts deployer address, an EOA set by the `PRIVATE_KEY` in env config.

Deployment scripts do not include an ownership transfer to a more secure owner, like a multisig. This increases the risk of a complete protocol hijack if the keys are compromised.

Recommendation

Consider transferring ownership of the `GlobalStorage` contract to a multisig address in deployment scripts.

Alternatively, add an address param to the `GlobalStorage` constructor, and use it to initialize the `Ownable` parent contract.

Resolution

Bounce Team: Acknowledged.

L-02 | Missing Token Support Validation

Category	Severity	Location	Status
Validation	Low	position_service.py: 162	Partially Resolved

Description

Although the `filter_tokens_by_leverage_constraints` function filters the unsupported leverage tokens, there are still some missing validation:

• No validation at config time; config validation passes, service starts, only at runtime the tokens are filtered

• Event queue mismatch; although `leverage_token_map.get` fetches the filtered list, the `_enqueue_event` uses `configs.token_supported.get_field`.

• Hyperliquid max leverage changes: If Hyperliquid reduces max leverage for an asset (e.g., HYPE goes from 10x to 8x), the system won't detect this until the service restarts or a config reload triggers.

Recommendation

Consider adding the supported token validation on leverage constraints whenever the script reads token list.

Resolution

Bounce Team: The issue was resolved in PR#96.

L-03 | Incorrect Redemption Execution Check

Category	Severity	Location	Status
Unexpected Behavior	Low	blockchain.py: 343	Resolved

Description

During the redemption execution flow, the scripts first check if the executeRedemptions call will revert with the user array length.

The executeRedemptions(users_checksum).call is the command used to check the transaction result, but it does not set a specific gas limit, so the node will use its own default value.

This leads to an invalid execution checks, as the check passes simulation but the estimated gas for the transaction can exceed the Hyperliquid fast block gas limit.

Although the flow uses _execute_with_halving_retry to reduce the user list and retry the transaction, it's something that could have been avoided in the previous checks (binary search)/.

During internal testing, if the list contains more than 27 executable users, it will exceed 3M gas.

Recommendation

Consider simulating the executeRedemptions call with a gas value equal to the fast block gas limit:

```
result = contract.functions.executeRedemptions(users_checksum).call({"from":
self._account.address, "gas": 3_000_000})
```

Resolution

Bounce Team: The issue was resolved in PR#95.

L-04 | No Core Settlement Verification

Category	Severity	Location	Status
Logical Error	Low	contract_to_spot.py: 47	Resolved

Description

The automation treats the EVM `bridgeIn` transaction as final. After it confirms, a Hyperliquid Core spot transfer is triggered, but the scripts do not verify Core settlement.

If the Core leg fails post-EVM (e.g., Core-side error or partial execution), the flow reports success and will not retry, leaving funds potentially stranded or state out of sync.

This issue may rise if a `bridgeIn` transaction is send in EVM, but there are insufficient funds in Core to send.

Recommendation

After EVM `bridgeIn` confirmation, add a post-check against Hyperliquid (e.g., verify spot balance/transfer status) and raise/retry if the Core transfer did not settle.

Alternatively, emit/log Core settlement status from the handler and require the automation to confirm it before marking success.

Resolution

Bounce Team: The issue was resolved in PR#107.

L-05 | Incomplete Private Key Cleanup Mechanism

Category	Severity	Location	Status
Censoring	Low	scripts/add-account.sh: 113	Resolved

Description

The "scripts/add-account.sh" script uses `trap ... RETURN.` for cleanup of the private key temporary files. However, this trap only executes if the function finishes its execution flow normally.

If the script is interrupted (e.g., Ctrl+C), killed by the CI/CD runner (timeout/cancellation), or encounters a syntax error causing a crash, the RETURN signal is never sent.

As a result, the temporary file containing the plaintext private key will remain in /tmp/ (or the temp directory) indefinitely, accessible to anyone with sufficient permissions on that machine.

Recommendation

Use `trap ... EXIT` (which covers script termination) or specifically trap signals like SIGINT and SIGTERM.

Resolution

Bounce Team: The issue was resolved in PR#97.

L-06 | Potential Exposure Of Private Key Through Logs

Category	Severity	Location	Status
Best Practices	Low	scripts/add-account.sh: 115	Resolved

Description

The line `echo -n "$private_key" > "$temp_key_file"` is insecure in CI/CD and debugging environments.

https://github.com/GuardianOrg/bounce-automation-team1-1761403919469/commit/38f80991eee3641c75cd5be639578f6564d6a78f#diff-65d3043d6a761a69290145f45d21970cf2ac6d4d0dd04fc069c713bf36e12328R115

If set -x (debug mode) is enabled anywhere in the script execution chain, the shell will expand the variable before running the command. The logs will literally show: `+ echo -n '0x3a1f...' > /tmp/tmp.XyZ`.

Thus, the private key is permanently recorded in the build logs, which are often viewable by a wider audience than the secrets manager itself.

Recommendation

Do not hold the key in a variable. If you must, use set +x before the command, or use cat with a heredoc (though the variable is still in memory).

Resolution

Bounce Team: The issue was resolved in PR#98.

L-07 | Potential Private Key Exposure Via Shell Memory

Category	Severity	Location	Status
Best Practices	Low	scripts/add-account.sh: 105	Resolved

Description

The architectural decision to handle the private key as a shell variable ($private_key) before writing it to a file negates the primary security benefit of file-based secret handling.

By accepting the key as a function argument (local private_key=$4), the secret is loaded into the shell process's memory. If the script crashes and generates a core dump, or if the environment is exported for debugging, the key is exposed in plaintext.

Depending on how the script invokes other commands, there is a risk that the variable could be implicitly passed to child processes or visible in the /proc/{pid}/environ file of the running shell (though local mitigates this, it does not eliminate the memory footprint).

The secret exists in the application layer for the entire duration of the variable's lifecycle, rather than only existing on disk with restricted permissions.

Recommendation

• Refactor the upstream logic (e.g., the Python script) to write the secret directly to a secure file.

• The shell script should only ever handle the file path string, never the secret content itself.

Resolution

Bounce Team: The issue was resolved in PR#106.

L-08 | Insecure File Creation Window (Race Condition)

Category	Severity	Location	Status
Unexpected Behavior	Low	109-114	Resolved

Description

There is a Time-of-Check to Time-of-Use (TOCTOU) style race condition between the creation of the temporary file and the application of secure permissions.

The Gap: `mktemp` creates the file using the current system umask (often 0022, making files world-readable by default). The script then runs chmod 600 in a subsequent command.

Between the execution of `mktemp` and `chmod`, there is a non-zero time window where the file exists on the filesystem with loose permissions.

A malicious actor or compromised process monitoring `/tmp` (using inotify or a busy-loop) could open a file handle to read the content the moment the file is created, effectively bypassing the subsequent `chmod`.

Recommendation

• Ensure atomic secure creation.
Option 1 (Umask): Set a strict umask immediately before creation:

```
old_umask=\$(umask)
umask 077
temp_key_file=\$(mktemp)
umask \$old_umask
```

Option 2: As recommended in earlier issue, let Python handle this using `os.open` with specific mode flags (0o600) to ensure the file is never created with insecure permissions.

Resolution

Bounce Team: The issue was resolved in PR#99.

L-09 | "Fail Open" On DNS Failure

Category	Severity	Location	Status
Error	Low	src/common/utils.py: 80	Resolved

Description

The code explicitly allows URLs to pass validation if DNS resolution fails. This negates the security control.

```
except socket.gaierror:
return True, None  # <--- VULNERABILITY
```

An attacker can provide an internal hostname that your specific validation environment can't resolve (or they can make their DNS server time out intentionally).

The check returns `True`. Later, the actual HTTP client (which might have different timeout settings or cached DNS entries) successfully resolves the internal address and performs the SSRF attack.

Recommendation

Security controls must Fail Closed. Instead of return `True`. `return False, "DNS resolution failed; cannot verify safety."`

Resolution

Bounce Team: The issue was resolved in PR#100.

L-10 | Potential DNS Rebinding Vulnerability (TOCTOU)

Category	Severity	Location	Status
Best Practices	Low	src/common/utils.py: 73	Acknowledged

Description

This implementation suffers from a classic Time-of-Check to Time-of-Use (TOCTOU) vulnerability.

You are validating the DNS record at a specific point in time, but you are not enforcing that the subsequent HTTP request uses that exact same IP address.

Time of check: If the code calls `socket.gethostbyname(hostname)`. The attacker's DNS server returns a safe IP (e.g., 8.8.8.8) with a TTL (Time To Live) of 0 seconds. The validation passes.

Gap: The code returns True. The application logic proceeds to make the actual RPC call (using requests, aiohttp, or `web3.py`).

Time of use: Time of Use: Because the TTL was 0, the HTTP client triggers a new DNS resolution. The attacker's DNS server detects the second query and now returns a private IP (e.g., 127.0.0.1 or 169.254.169.254).

As a result, the HTTP client connects to the internal resource, bypassing your security check entirely.

Recommendation

• Do not rely on the hostname for the actual connection.

• Resolve the IP address.

• Validate the IP address.

• Force the connection to the IP: Construct the URL using the validated IP address (e.g., https://1.2.3.4/rpc) and manually set the Host header to the original hostname so the server accepts the request (SNI/Virtual Host compatibility).

Resolution

Bounce Team: Acknowledged.

L-11 | Thread-Unsafe Global State Mutation

Category	Severity	Location	Status
Best Practices	● Low	src/common/utils.py: 70-75	Resolved

Description

The function modifies socket.setdefaulttimeout(), which is a process-wide global setting in Python.

```
original_timeout = socket.getdefaulttimeout()
try:
socket.setdefaulttimeout(5)  # <--- Global lock on all new sockets
finally:
socket.setdefaulttimeout(original_timeout)
```

In a multi-threaded application (common in frameworks handling RPCs, database connections, or API endpoints):

Interference: If Thread A is running this validation, it sets the global timeout to 5 seconds.

Collateral Damage: If Thread B starts a database connection, a Redis operation, or a long-polling request at that exact moment, it inherits the 5-second timeout. This can cause random TimeoutError exceptions in completely unrelated parts of your application that expect different timeout behaviors (or no timeout at all).

Race Conditions: If two threads run this validation function simultaneously, they will fight over the original_timeout value, potentially permanently leaving the application in a state with a 5-second global timeout.

Recommendation

Never change global defaults for local operations. Use socket.create_connection (which accepts a timeout argument) or generic DNS libraries that allow context-specific timeouts.

Resolution

Bounce Team: The issue was resolved in PR#101.

L-12 | IPv6 Validation Bypass

Category	Severity	Location	Status
Logical Error	Low	src/common/utils.py: 73	Resolved

Description

The code relies exclusively on `socket.gethostbyname()`, which is a legacy function that typically resolves only IPv4 addresses (A records).

The Mechanism:

• Modern operating systems and networking libraries (like requests or `urllib3`) prefer IPv6 over IPv4 ("Happy Eyeballs" algorithm).

• If an attacker configures a domain dualstack.evil.com with:
`A Record (IPv4): 1.1.1.1 (Public/Safe)`
`AAAA Record (IPv6): ::1 (Localhost/Unsafe)`

The Bypass:

• The validation calls gethostbyname, sees 1.1.1.1, and returns Valid.

• The HTTP client performs the request, prefers the AAAA record, and connects to ::1.

• The request hits your local internal network.

Recommendation

Use `socket.getaddrinfo` instead of `gethostbyname`. You must iterate through every returned IP address (both `AF_INET` for IPv4 and `AF_INET6` for IPv6) and ensure all of them are non-private. If even one resolution points to a private network, the URL must be rejected.

Resolution

Bounce Team: The issue was resolved in PR#101.

L-13 | Flawed Deduplication Logic In Telegram Alerting

Category	Severity	Location	Status
Logical Error	Low	src/services/alerting_service.py: 108	Resolved

Description

The `_is_duplicate_alert` method determines uniqueness based solely on metadata (`process_name`, `instance_name`, `logger_name`, and contexts). It explicitly ignores the message (body) of the alert.

The service will silently drop critical alerts. If a specific logger emits a generic error (e.g., "Connection Start"), followed immediately by a critical error (e.g., "Critical Data Corruption"), the second error will be discarded as a duplicate because the metadata context is identical.

Recommendation

Include a hash of the message body in the deduplication comparison.

Resolution

Bounce Team: The issue was resolved in PR#102.

I-01 | Unnecessary Gas Spent For Executions

Category	Severity	Location	Status
Gas Optimization	Info	blockchain.py: 337	Resolved

Description

When executing pending redemptions, the service now verifies if at least one user redemption in the list will be executable. Although the transaction will succeed, it will waste unnecessary gas for the non-executable users when looping through all user list.

Recommendation

Consider filtering the redemptions that will be executed, using the simulation inside `can_execute_redemptions`.

Resolution

Bounce Team: The issue was resolved in PR#104.

I-02 | Unsupported Leveraged Tokens

Category	Severity	Location	Status
Informational	Info	leverage_utils.py: 34	Acknowledged

Description

The `PerpAssetConfig` now verifies if:

1 - position to open above a % of the Hyperliquid max perp leverage (currently 90%)

2 - token leverage above max TVL leverage (6.3x)

However, the factory contract does not enforce any of this. If an unsupported token is deployed (i.e. SOL10L), users can mint LTs but the automation scripts will never support it.

Keep in mind that tokens like SOL and HYPE have a perps max leverage of 10x, so they are right at the limit.

Recommendation

Consider either documenting these validations in the `Factory` contract or enforcing it via config params during `_validateLeverage`.

Resolution

Bounce Team: Acknowledged.

I-03 | Deployment Script Private Key Handling

Category	Severity	Location	Status
Best Practices	Info	DeploymentScript.sol: 20	Resolved

Description

Deployment script loads the private key via raw `PRIVATE_KEY` environment variable. Raw private keys are easily leaked through shell history, process lists, CI logs, or accidentally committed `.env` files.

Recommendation

Switch to Foundry's encrypted keystore more info:

https://getfoundry.sh/guides/best-practices/key-management/#using-a-keystore

Resolution

Bounce Team: The issue was resolved in PR#151.

I-04 | Insecure Private Key Handling

Category	Severity	Location	Status
Best Practices	Info	add-account.sh: 148	Resolved

Description

The script stores the generated private key in shell variables, writes it to .env, and passes it as a command-line argument. This leaks the key via ps aux, shell history, logs, and version control.

Recommendation

Never let the private key touch the shell. Generate and upload it directly from Python (or use Foundry/cast keystore), return only the address, and store just the address/reference in .env and YAML files.

Resolution

Bounce Team: The issue was resolved in PR#106.

I-05 | Gaming The Order Of The Redemption Queue

Category	Severity	Location	Status
Logical Error	Info	LeveragedToken.sol: 416	Acknowledged

Description

`executeRedemptions()` processes `pendingRedemptions` array sequentially. In `_removeCredit()`, swap-and-pop removal allows attackers to front-run executions:

by timing a new redemption as the last element, popping shifts them to an earlier position (e.g., front), enabling order manipulation.

Recommendation

Replace with order-preserving queue handling/Execute the redemptions ordered by the `pendingRedemptionTimestamp` set during prepare

Resolution

Bounce Team: Acknowledged.

I-06 | Potential Info Disclosure Through Telegram API

Category	Severity	Location	Status
Best Practices	Info	src/services/alerting_service.py: 166, src/services/alerting_service.py: 189	Resolved

Description

The service blindly forwards error logs to Telegram without inspecting the content for sensitive information.

Developers often log exceptions that include full context. For example, a database connection error might include the connection string (with password), or an API error might dump the request headers (with Bearer tokens).

An attacker who gains access to the Telegram channel (or the Telegram servers' history) can view these secrets.

Recommendation

Implement a "Redactor" or "Sanitizer" utility that regex-matches and masks patterns like `API_KEY`, `Bearer`, `password=`, and private key formats before the message is formatted.

Resolution

Bounce Team: The issue was resolved in PR#105.

I-07 | Missing API Rate Limiting In Telegram Alerting

Category	Severity	Location	Status
Best Practices	Info	/src/services/alerting_service.py: 189	Resolved

Description

The service does not implement client-side rate limiting. Telegram enforces strict limits (~30 msgs/sec globally, ~1 msg/sec per chat).

During an error burst (e.g., database downtime), the service will flood the API, receive `429 Too Many Requests` errors, and likely throw exceptions in the handler. This can trigger an infinite feedback loop where the Alerting Service logs errors about failing to send errors.

Recommendation

Implement a "Leaky Bucket" rate limiter or a "Circuit Breaker" to pause notifications when limits are reached.

Resolution

Bounce Team: The issue was resolved in PR#103.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits