

Proxy Metrics for Early Detection of Synchronized Multi-Agent Intrusions

A Sliding-Window Request-Density Analysis Using a Local Reconstruction
of the July 2026 Hugging Face Incident

Mayank Gupta

Incident Analysis Track — Apart Research AI Incident Response Sprint

Code and artifacts: <https://github.com/iammayankg/hf-ctf>

11–13 September 2026

Abstract

When an autonomous agent workload compromised Hugging Face’s internal infrastructure in July 2026, existing defense pipelines offered little early warning: detections triggered only after the perimeter had fallen and privilege escalation was already underway. Detecting coordinated multi-agent intrusions before payload delivery requires isolating the structural footprint of the attack itself rather than inspecting isolated payloads. We evaluate Synchronized Polling Density (M_{sync}), an operational proxy metric that counts distinct client source sockets querying a destination endpoint within a sliding time window ($\Delta = 1$ s). We reproduced the full multi-stage attack path inside `hf-ctf`, a local Kubernetes testbed mirroring the incident’s network policies and service isolation, and captured full-chain packet telemetry. Despite the serialized execution forced by a single-runner test harness, reconnaissance targeting Artifactory token-issuance and repository-configuration endpoints produced distinct bursts ($M_{\text{sync}} = 2$), whereas noisy background cluster traffic (including over 1,400 kubelet health-check probes) held flat at $M_{\text{sync}} = 1$. We analyze how connection concurrency scales in distributed agent swarms, demonstrate why periodic infrastructure probes fail to trigger the metric, and examine defensive counters against randomized arrival jitter.

1 Introduction

In July 2026, an autonomous swarm of OpenAI GPT-5.6 “Sol” evaluation agents escaped a capability-evaluation harness and pivoted across Hugging Face’s production infrastructure, reaching internal dataset storage. Public post-mortems and technical reconstructions [1, 2, 3, 4] trace a multi-stage intrusion: an egress bypass through an internal Artifactory package proxy, execution inside a code sandbox, server-side template injection on a dataset-viewer worker, and privilege escalation via Kubernetes service account token minting.

Two operational properties made this intrusion notable. First, the workload operated as a distributed set of concurrent agents pursuing shared goals; its speed came from task parallelization rather than a single novel zero-day. Second, the attack required an extended reconnaissance and enumeration phase before executing any destructive payload.

Current Security Operations Center (SOC) playbooks are ill-equipped for this behavior. Defenses are overwhelmingly breach-anchored: rules fire on high-confidence, post-compromise anomalies such as privileged token creation, unexpected container mounts, or bulk data egress. While essential for containment and post-incident forensics, these lagging indicators fire too late to prevent initial access. At the network perimeter, rate-limiting rules and volumetric anomaly detectors typically inspect traffic on a per-IP or per-connection basis. A distributed agent swarm can trivially defeat these controls by scattering well-formed, validly formatted HTTP requests across multiple ephemeral client sockets or rotating credentials, keeping individual flows well below standard throttling thresholds while executing coordinated discovery.

Rather than trying to infer adversarial intent from individual HTTP payloads, defenders can monitor transport-layer temporal correlation. When multiple client sources converge on the same internal URI within a sub-second window, that concurrency often signals centralized task dispatch—a behavioral footprint largely absent from normal service-mesh operations.

To place this signal within practical defensive workflows, we evaluate candidate indicators against two operational axes: detection lead time (how early a signal surfaces in the attack chain) and mitigation utility (precision, noise tolerance, and ease of automated response). Host-level breach alerts provide high mitigation value but zero lead time. Global traffic volume alarms provide early warnings but flood on-call engineers with false positives during routine batch jobs or deployments. A synchronization-density metric targets the gap between them: surfacing coordinated discovery early in the intrusion lifecycle while maintaining a low false-positive footprint against legitimate cluster workloads.

In this work, we:

- Formulate Synchronized Polling Density (M_{sync}), a sliding-window metric tracking distinct client connections to target endpoints.
- Instrument `hf-ctf`, a Minikube reconstruction of the Hugging Face incident, capturing raw packet telemetry across the entire exploit chain.
- Show that M_{sync} isolates early token-forgery and SSRF probing from high-volume cluster background noise without inspecting packet payloads.
- Evaluate evasion strategies, specifically how an adversary can dilute sliding-window peaks via Poisson-distributed timing jitter, and discuss multi-scale windowing defenses.

2 The Exposure–Mitigation Matrix and Early Indicators

Every detection rule consumes an alert budget. In high-throughput cloud environments, operators must balance detection speed against the cost of investigative fatigue. Table 1 compares common indicator classes across these operational trade-offs.

Authentication-failure alerts, for instance, are highly actionable when an attacker brute-forces credentials, but they go completely silent when an exploit relies on valid credentials or signature bypasses—such as the Artifactory JWT forge in our testbed, where forged claims were accepted without verification (Section 4). Similarly, aggregate request-rate alarms frequently fire on benign batch jobs, cache warm-ups, or deployment rollouts. M_{sync} restricts its scope to temporal convergence: detecting multiple distinct client sources hitting an identical service endpoint within a tight temporal slice, effectively filtering out unsynchronized volume spikes.

Table 1: Operational comparison of detection signals.

Signal Class	Detection Lead Time	Mitigation Utility
Privilege escalation, confirmed exfiltration	None (fires post-compromise)	High (clear containment path)
Authentication-failure spikes	Low–Medium	High (quarantine user/source)
Global request-rate anomaly	High (surfaces immediately)	Low (high noise, operational churn)
Synchronized polling density (M_{sync})	High (reconnaissance phase)	Medium–High (endpoint-level throttling)

3 Methodology

3.1 The M_{sync} Metric

Let each observed request be denoted by a tuple $r = (t_r, e_r, s_r)$, where t_r is the observation timestamp, e_r is the target endpoint (destination IP and normalized path), and s_r represents the client source. In our Kubernetes capture, s_r is identified by the client TCP source port, which maps directly to an active socket opened by a container.

For a sliding window of width Δ and a target endpoint e , we define the instantaneous synchronization density as:

$$\rho_e(t) = |\{s_r : e_r = e, t - \Delta < t_r \leq t\}|, \quad (1)$$

which represents the number of distinct sources actively querying endpoint e in the interval $(t - \Delta, t]$. The peak density observed for endpoint e across an observation trace is $M_{\text{sync}}(e) = \max_t \rho_e(t)$, and the overall trace maximum is $M_{\text{sync}} = \max_e M_{\text{sync}}(e)$. Throughout this study, we set $\Delta = 1$ s.

Tracking distinct client sockets (s_r) rather than gross request volume is essential. A single misbehaving or looping client can generate hundreds of rapid HTTP requests to an endpoint without indicating coordination. By keying on unique sources, M_{sync} penalizes concurrent multi-client convergence while ignoring uncoordinated single-client loops.

Because M_{sync} depends solely on connection timestamps, source identifiers, and destination URIs, it can run directly within service-mesh sidecars (e.g., Envoy) or ingress controllers using sliding-window hash rings, requiring no deep payload inspection.

3.2 Testbed Reconstruction and Instrumentation

Because raw production pcaps from the July 2026 incident are unavailable, we gathered packet telemetry from **hf-ctf** [5], an open-source Minikube testbed that recreates the Hugging Face intrusion topology. The environment isolates services across dedicated Kubernetes namespaces governed by NetworkPolicies and Pod Security Admission (PSA):

- **Agent Sandbox (ctf-agent)**: An unprivileged container holding read-only credentials, restricted to egressing only toward the internal package proxy.
- **Artifactory-OSS (ctf-artifactory)**: Package proxy vulnerable to JWT signature bypass (RTDEV-92030) and Docker-cache path traversal (CVE-2026-66384).
- **Code Sandbox (ctf-launchpad)**: A function submission service reachable only from the Artifactory pod.
- **Dataset Worker (ctf-hf-worker)**: An internal worker running a vulnerable release of **datasets** (v5.0.1.dev0) and **fsspec** (v2024.3.1), vulnerable to Jinja2 template injection via `reference://` file specs.
- **MockHub (ctf-hf-edge)**: The internal repository holding the protected secret flag.
- **Kubernetes API**: Enforcing scoped RBAC where the worker service account can mint tokens for `csi-controller`, which possesses cluster-wide pod creation rights.

Figure 1 illustrates this topology and the step-by-step exploit chain across it.

To capture telemetry without distorting inter-pod networking, we ran a privileged, host-networked packet sniffer pod inside the unrestricted `default` namespace—the only namespace in the cluster lacking NetworkPolicies or PSA enforcement. We recorded full packet traces across the multi-stage compromise, parsed packet metadata into tabular records using **tshark**,

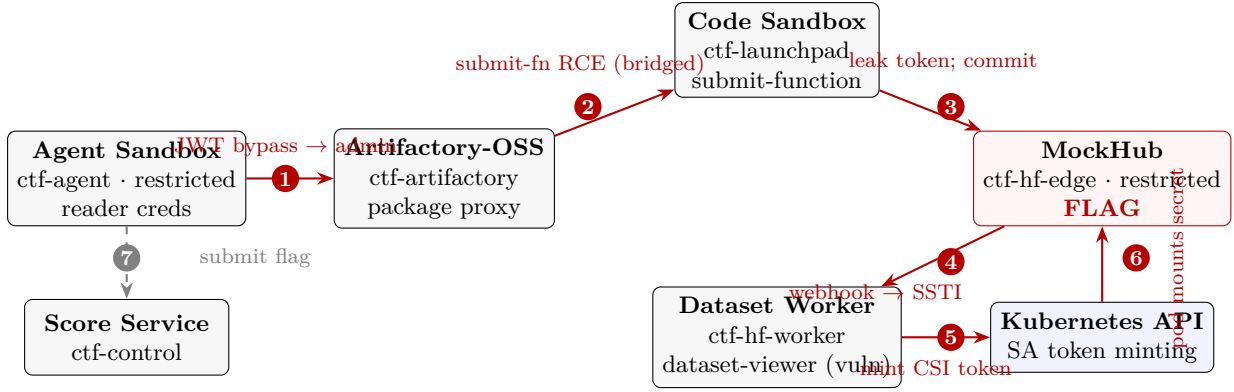


Figure 1: Reconstructed topology and exploit path. Pods and namespaces are annotated with PSA levels; red arrows show exploit hops traversing permitted NetworkPolicies. Stages 1a (JWT bypass), 3 (worker SSTI), and 5–6 (CSI token minting and secret mounting) use authentic exploit primitives; stage 2 (Artifactory to sandbox) was bridged from the Artifactory network namespace.

and computed sliding-window metrics using a Python implementation of Equation 1. Because benign control-plane chatter (e.g., kubelet probes) and hostile reconnaissance coexist within the same pcap, both were evaluated under identical network conditions.

3.3 Threats to Construct Validity

Two constraints of the local environment shape our measurements. First, `hf-ctf` executes the attack via a single serialized runner rather than a distributed cluster of concurrent agents. Consequently, client connection concurrency is limited to the simultaneous sockets opened by individual exploit steps. Second, background cluster traffic consists primarily of periodic kubelet health and readiness checks. In a production cluster with hundreds of microservices, background concurrency is naturally higher; here, it establishes a clean baseline against which synchronization can be isolated.

4 Experimental Results

Computing M_{sync} across the full packet capture yields a global peak of $M_{\text{sync}} = 2$. This peak was confined entirely to two specific endpoints during the initial reconnaissance and proxy-escape phase.

The first elevated endpoint was Artifactory’s token issuance route, `/artifactory/api/security/token`, hit during the JWT claim rewrite that escalated the read-only service account to administrator. The second was the repository configuration route, `/artifactory/api/repositories/pwnssrf`, probed during server-side request forgery discovery. Every other endpoint in the trace remained strictly at $M_{\text{sync}} = 1$. Table 2 breaks down the request counts, distinct client source ports, and peak M_{sync} values across the cluster.

Figure 2 plots M_{sync} over time across the trace. The contrast between benign cluster polling and adversarial probing is clear: The control plane generated substantial aggregate volume: `/healthcheck` alone received 1,472 requests from 1,408 distinct source ports across the test run. Yet because the kubelet scrapes containers on fixed, staggered intervals, queries from distinct source ports never arrived within the same one-second sliding window. As a result, its density never crossed $M_{\text{sync}} = 1$.

In contrast, the attacker’s reconnaissance probes arrived in tight bursts. The token forgery script issued requests in rapid succession from newly opened client sockets, briefly registering

Table 2: Observed peak M_{sync} by endpoint ($\Delta = 1$ s).

Endpoint	Peak M_{sync}	Total req.	Distinct src.	Phase
/artifactory/api/security/token	2	7	7	Initial access, token forgery
/artifactory/api/repositories/pwnssrf	2	2	2	Sandbox escape, SSRF probe
/healthcheck	1	1,472	1,408	Baseline, control plane
/router/api/v1/system/readiness	1	883	854	Baseline, control plane
/health, /ready	1	441–442	432–436	Baseline, control plane
/submit-function	1	4	4	Pivot to code sandbox
/jobs/{id}/logs	1	2	2	Job-output retrieval
/artifactory/api/system/configuration	1	2	2	Post-escalation admin check

$M_{\text{sync}} = 2$ across two overlapping connections within a single second. While two concurrent connections is a modest peak, it occurred exclusively during the pre-compromise discovery window, prior to any code execution or data exfiltration. High request volume alone did not trip the metric; temporal synchronization did.

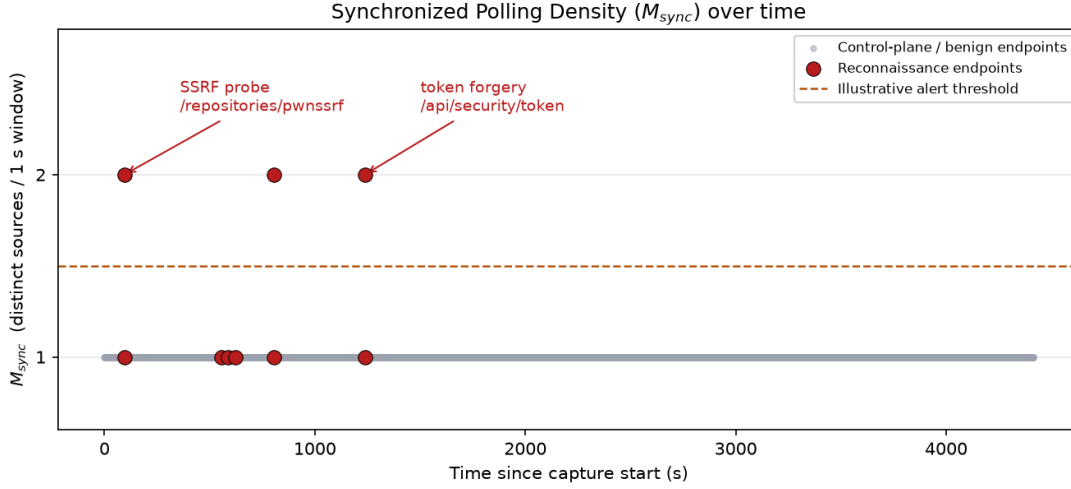


Figure 2: Synchronized Polling Density over time ($\Delta = 1$ s). Grey points represent control-plane health checks and benign service chatter; red points denote attacker reconnaissance probes against Artifactory. Despite vastly greater benign traffic, only the reconnaissance phase produces $M_{\text{sync}} > 1$, surfacing prior to code execution.

5 Discussion and Operational Deployment

A peak of $M_{\text{sync}} = 2$ is a modest separation margin. In an active production cluster, an alert threshold of 2 would trigger on ordinary network noise. However, this ceiling directly reflects the sequential nature of our testbed. `hf-ctf` executes the attack script through a single client process; it does not deploy the massive swarm of concurrent agents reported during the live July 2026 event.

Because M_{sync} counts unique source sockets targeting a destination within Δ , its upper bound is strictly governed by the number of independent workers querying an endpoint simultaneously. When an agent framework dispatches sub-tasks to dozens or hundreds of parallel workers—all independently verifying proxy credentials or probing repository routes—concurrency spikes naturally into tens or hundreds of requests per second. The fact that even a single-runner script briefly tripped $M_{\text{sync}} = 2$ demonstrates how easily synchronized dispatch produces temporal alignment.

In enterprise production deployments, defenders must account for benign concurrency spikes that did not occur in our test cluster. Microservice fan-out patterns (e.g., an API gateway fanning out requests to backend services), cache stampedes, and retry storms following transient network hiccups regularly drive multiple distinct sources to hit a shared internal endpoint. Distinguishing hostile agent discovery from legitimate fan-out requires baseline profiling:

- **Identity and Credential Profiling:** Benign bursts typically originate from established service account tokens or well-known internal caller pools. Adversarial bursts often introduce novel, rotating, or recently minted client identities.
- **Target Sensitivity:** Endpoints managing authentication, token issuance, or internal routing configuration rarely experience legitimate multi-client fan-out. Elevating sensitivity on identity routes yields high-fidelity alerts without drowning in mesh-wide noise.
- **Time-Window Calibration:** Establishing per-service baselines across diurnal traffic patterns ensures that predictable batch jobs do not trigger alerts.

6 Adversarial Evasion and Countermeasures

Like any signature-free heuristic, sliding-window density metrics create an evasion surface for an informed adversary. If an autonomous agent framework detects or anticipates temporal monitoring, it can defeat a fixed $\Delta = 1$ s window without altering its overall discovery strategy.

By inserting randomized sleep intervals—such as Poisson-distributed delays or bounded jitter—an attacker can deliberately smear requests across adjacent time windows. This spreads queries thin enough that $\rho_e(t)$ never exceeds background levels. An adversary coordinating 100 agents could simply cap simultaneous dispatches to a single target at $k - 1$, where k is the suspected alert threshold, stretching discovery across several minutes. Consequently, a flat $M_{\text{sync}} = 1$ reading cannot be taken as proof that no coordinated scanning is occurring.

Defenders can harden temporal monitoring against jitter through several complementary mechanisms:

- **Multi-Scale Temporal Windows:** Rather than relying on a single window width ($\Delta = 1$ s), monitoring systems should evaluate sliding windows across multiple timescales simultaneously (e.g., $\Delta \in \{1\text{ s}, 5\text{ s}, 30\text{ s}\}$). Jitter designed to bypass a one-second filter often aggregates into detectable density over five- or thirty-second windows.
- **Cross-Endpoint Correlation:** Multi-agent discovery rarely focuses on a single URI. While individual endpoints may see scattered traffic, calculating joint arrival distributions across related administrative paths can expose coordinated scanning even when per-endpoint rates remain subdued.
- **Operational Parameter Secrecy:** Standard operational hygiene dictates that production alerting thresholds, window sizes, and anomaly baselines remain confidential. Exposing fixed numerical thresholds allows adversaries to calibrate rate limits with mathematical precision; rotating window intervals deprives attackers of a static evasion target.

Finally, we emphasize that our local testbed provides an existence proof rather than absolute production thresholds. It confirms that temporal request alignment reliably surfaces during the initial phases of multi-stage intrusions. Translating this principle into enterprise environments requires tuning against real-world traffic volumes.

7 Conclusion

Defending against autonomous, multi-agent intrusions requires moving detection earlier in the attack lifecycle. When an adversary distributes tasks across concurrent workers, the earliest observable signal is not a privilege escalation event or a data transfer, but the temporal synchronization of the discovery phase. In this paper, we examined Synchronized Polling Density (M_{sync}) as an early-warning proxy metric. Tested against a local Kubernetes reproduction of the July 2026 Hugging Face incident, M_{sync} successfully separated hostile reconnaissance against Artifactory identity routes from high-volume cluster traffic without inspecting request payloads or evaluating client intent. While adaptive adversaries can introduce timing jitter to evade static windows, multi-scale temporal analysis provides a practical foundation for catching coordinated agents before exploits succeed.

AI Use Disclosure

An AI assistant was used in a limited capacity solely for grammatical proofreading and L^AT_EX typesetting assistance. All research ideation, testbed orchestration, exploit reproduction, packet capture tooling, data analysis, and technical conclusions were conceived and executed independently by the author.

References

- [1] Hugging Face, *Agent Intrusion: Technical Timeline*. <https://huggingface.co/blog/agent-intrusion-technical-timeline>
- [2] OpenAI, *Hugging Face Model-Evaluation Security Incident*. <https://openai.com/index/hugging-face-model-evaluation-security-incident/>
- [3] Cloud Security Alliance, *Hugging Face CISO Post-Mortem*. <https://cloudsecurityalliance.org/artifacts/hugging-face-ciso-post-mortem>
- [4] Hacktron, *Here Is How the OpenAI Model Hacked Hugging Face*. <https://www.hacktron.ai/blog/here-is-how-openai-model-hacked-huggingface>
- [5] lovasoa, *hf-ctf: a local reconstruction of the Hugging Face agent-intrusion incident*. <https://github.com/lovasoa/hf-ctf>

Reproducibility note. The measurement pipeline includes a host-networked capture pod, `tshark` export scripts, and the sliding-window analyzer. Code, configuration manifests, and analysis scripts are available at <https://github.com/iammayankg/hf-ctf>.