Detecting Piecewise Cyber Espionage in Model APIs

Arthur Collé
Distributed, Systems & Co.
arthur@distributed.systems

Alexander Reinthal
Independent
email@reinthal.me

Yingquan Li
STScI / NASA
yli12313@vt.edu

Linh Le
Mila / McGill University
thai.linh.le@mila.quebec

David Williams-King
ERA
david@erafellowship.org

Abstract

On November 13th 2025, Anthropic published a report on an AI-orchestrated cyber espionage campaign. Threat actors used various techniques to circumvent model safeguards and used Claude Code with agentic scaffolding to automate large parts of their campaigns. Specifically, threat actors split campaigns into subtasks that in isolation appeared benign. It is therefore important to find methods to protect against such attacks to ensure that misuse of AI in the cyber domain can be minimized. To address this, we propose a novel method for detecting malicious activity in model APIs across piecewise benign requests. We simulated malicious campaigns using an agentic red-team scaffold similar to what was described in the reported attack. We demonstrate that individual attacks are blocked by simple guardrails using Llama Guard 3. Then, we demonstrate that by splitting up the attack, this bypasses the guardrail. For the attacks that get through, a classifier model is introduced that detects those attacks, and a statistical validation of the detections is provided. This result paves the way to detect future automated attacks of this kind.

1 Introduction

The rapid advancement of large language models (LLMs) has created novel cybersecurity misuse attack vectors. Recently, Anthropic reported (Anthropic, 2025) the first documented case of large-scale AI-orchestrated cyber espionage. One month before the publication of the report, i.e. in October 2025, a threat actor leveraged Claude Code to conduct sophisticated attacks with minimal human oversight. Reconnaissance, exploitation, and data exfiltration were performed almost fully autonomously (80-90%). The core technique was simple: breaking a large malicious attack chain into discrete, seemingly benign sub-tasks. The sub-tasks would individually bypass AI safety guardrails, but collectively the attacker could carry out malicious operations at scale.

This mechanism of using piecewise benign requests—which we term a *benign-decomposition attack*—reveals a critical gap in current AI guardrail architectures. Traditional content moderation systems evaluate individual sessions in isolation, with safety built into models through fine-tuning, or through external guardrail models that run on inputs and outputs. However, splitting the tasks into pieces allows each task to appear benign, and the large (malicious) picture is outside the visibility of the individual monitors. As observers are noting (Jones, 2025), the security architecture of model APIs has to change fundamentally to block this style of misuse.

This paper presents a novel detection framework for identifying AI-orchestrated benign-decomposition attacks. Such attacks can leverage multiple user accounts, independent chat sessions, and role-play to make each request appear benign. Hence, relying on temporal relationships, or metadata like account name or specific IP addresses, is unreliable. Our method instead relies on categorizing each tool call as one of several attack stages in the cyber kill chain (Yadav and Rao, 2015). Then, we use correlations in the data such as domain names or target IP addresses mentioned across multiple stages to piece together a full attack.

We demonstrate our methodology through a simulated attack environment replicating four stages observed in the original real-world incident: *reconnaissance*, *vulnerability scanning*, *exploitation*, and *exfiltration*. We demonstrate that fully-specified attacks are blocked by simple guardrails. Then, we demonstrate that by splitting up the attack, it bypasses the guardrail. However, our trained classifier is able to categorize each piece and we are able to detect the end-to-end attack. See Figure 5 for a visualization of our method.

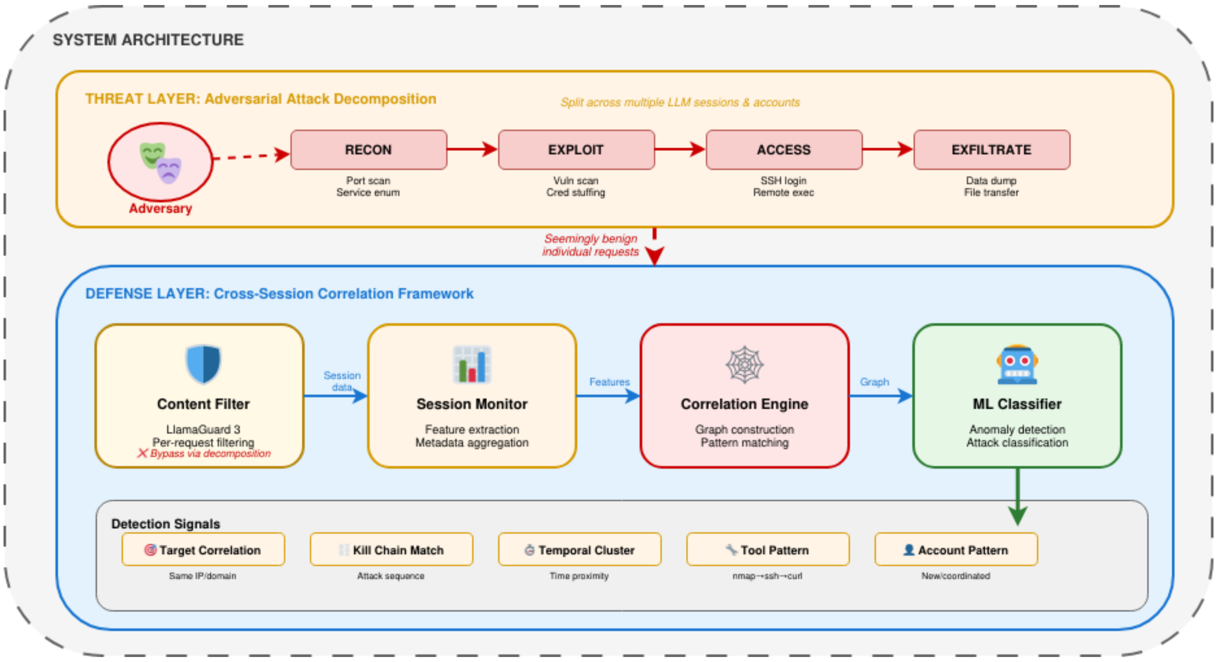


Figure 1: Diagram of our multi-layer system architecture. In the first layer, we describe different stages of the cyber kill chain that we will detect. In the defense layer, we rely not just on per-session guardrails, but also search for malicious activity across sessions. Detection involves positive matches in multiple stages of the cyber kill chain.

Our contributions include:

1. We propose a novel method for detecting benign-decomposition attacks in model APIs, by categorizing tool calls according to the cyber kill chain, and then linking steps together to find complete attacks.

2. We create an emulation of a commercial model API using open-weight models, including an MCP server (Hou et al., 2025) implementation, which can run guardrails of our choosing for research purposes.

3. We create an agentic red-team scaffold based on the attack described by Anthropic, which has both real execution and test modes for responsible research.

2 Related Work

Recently, Anthropic reported (Anthropic, 2025) the first documented case of large-scale AI-orchestrated cyber espionage, suspected to be carried out by a Chinese group dubbed GTG-1002. Reconnaissance, exploitation, and data exfiltration were all performed almost fully autonomously (80-90%), with human intervention only for high-level decisions or to direct the attack to new targets. The threat actor bypassed individual model guardrails by establishing a false persona as a “cybersecurity firm,” and tasking AI instances to perform narrowly-scoped operations—port scanning, credential validation, database queries—that only reveal malicious intent when analyzed as coordinated sequences. In the documented incident, attackers targeted approximately 30 organizations simultaneously and successfully compromised major technology corporations and/or government agencies.

MCP or Model Context Protocol (Hou et al., 2025) is a standard for exposing actions and tools to AI agents. It has been adopted by OpenAI, Anthropic, and other model providers. MCP enables agents to run programs including network scanning, remote login, etc., and was utilized by the cyber espionage reported by Anthropic. We describe our MCP server implementation in Section 4.1.

The cyber kill chain (Yadav and Rao, 2015) describes the stages that an attack goes through, from initial reconnaissance to exploitation to command-and-control, etc. The cyber kill chain inspired our classification categories, shown in Figure 5.

3 System Design/Architecture

3.1 Attack Model

Our cyber defense system (see Figure 5) implements a multi-layered architecture designed to detect sophisticated cyberattack misuse that spans multiple LLM sessions and user accounts. The architecture comprises two primary layers: 1) a threat model layer decomposing adversarial attack patterns, and 2) a defense layer employing a cross-session correlation framework.

The threat decomposition is based on the cyber kill chain. Specifically, we focus on the following categories:

1. Recon: Reconnaissance such as port scanning to look for attack surface.
2. Exploit: Using vulnerability scanning to look for ways into the system.
3. Access: Unauthorized breach such as logging into servers (ssh login).
4. Exfil: Exfiltrate data from the target (data dump), which is one of several attack payloads the attacker may care about.

3.2 Defense Architecture

Our system operates as a four-stage pipeline:

Content Filter Employing Llama Guard 3 for per-request filtering, this component serves as the initial defense layer. However, it can be bypassed through decomposition attacks where malicious intent is fragmented across multiple benign-appearing requests.

Session Monitor This component performs feature extraction and metadata aggregation from individual sessions, capturing behavioral patterns, request characteristics, and temporal information for downstream analysis.

Correlation Engine The core analytical component constructs graph-based representations of user interactions and employs pattern matching to identify relationships across sessions. It detects five attack signatures: target correlation (same IP/domain), kill chain matches (sequential attack stages), temporal clusters (time-proximate activities), tool patterns (nmap, ssh, curl usage), and account patterns (coordinated new accounts).

ML Classifier The final decision layer applies anomaly detection and attack classification algorithms, outputting definitive threat assessments based on correlated signals.

4 Implementation

We create an emulation of a commercial model API environment for test purposes; see Section 4.1. We also create an agentic red-team scaffold to model the attacker, which has both real execution and test modes for responsible research; see Section 4.2.

4.1 Model API Emulation

Our defensive gateway (HACKERFINDER9000) implements an OpenAI-compatible API (`/v1/chat/completions`, `/v1/models`) using FastAPI. It operates as a transparent proxy, forwarding requests to a local vLLM server while analyzing traffic in real-time. The gateway supports configurable blocking (threshold-based at 0.8) or monitoring-only modes.

MCP Protocol Implementation MCP (Model Context Protocol) uses stdio-based communication between the client and tool servers. We built 13 MCP servers as standalone Python processes using the FastMCP library. Each server exposes tools via JSON schemas that are converted to OpenAI-compatible function calling format. The communication flow is: (1) client spawns server subprocess via `StdioServerParameters`, (2) client sends `initialize` to establish protocol, (3) client calls `tools/list` to discover available tools, (4) for each tool invocation, client sends `tools/call` with arguments, (5) server executes and returns results via `CallToolResult`.

Tool Registry A centralized registry (`mcp_registry.py`) manages all servers with: lazy loading (servers spawn only when needed), a 12K token context budget with automatic eviction, thermal context management (8-tier priority from PLASMA to ARCTIC based on recency/frequency), and tool namespacing to avoid collisions (e.g., `network_port_scan`).

Test Mode All servers support test mode via `MCP_TEST_MODE=true`, returning realistic fake responses without executing actual operations. For example, `fake_port_scan()` returns probabilistic open ports (10% per port) with proper service names and nmap-style output formatting. This enables safe dataset generation.

4.2 Red-Team Scaffolding

The red-team scaffold (RED-APT) bridges MCP servers to vLLM, enabling autonomous multi-stage attack chains that replicate the capabilities described in the Anthropic report.

MCPvLLMClient The core client (795 lines) implements: multi-turn tool execution with configurable iteration limits (default 10), refusal detection using 8 regex patterns, system prompt mutation (prepend/append/replace/wrap modes), and baseline labeling via HTTP headers (`X-Baseline-Type: benign|malicious`, `X-Task-Category`).

Attack Chain Orchestration We implemented baseline scripts for complete attack chains: (1) *Credential stuffing*: fetch credentials → SSH brute-force → C2 connect → file discovery → exfiltrate; (2) *APT simulation*: recon → vuln discovery → exploitation → persistence → exfil.

Infrastructure The system runs on Lambda Labs (8× A100 40GB): vLLM serving Qwen3-VL-30B-abliterated with 8-way tensor parallelism, FastAPI gateway on port 8000, PostgreSQL via Docker for kill chain database.

5 Results

Setup We used an abliterated version of Qwen3-VL 30B[1] for all our experiments. We used a Lambda Labs server with 8× A100 40GB's as our shared work environment. We thank Apart Research for the opportunity to present at the Defense Acceleration Hackathon 2025 and Lambda Labs for their generous contribution of compute credits.

Guardrail Bypass Demonstration Complete attack requests (e.g., "scan network 192.168.1.0/24, brute-force SSH, exfiltrate /etc/passwd") are correctly blocked by Llama Guard 3. However, when decomposed: "list open ports on 192.168.1.0/24" (network diagnostics), "test SSH password authentication" (sysadmin task), "read /etc/passwd" (file access)—each typically passes individually, demonstrating the benign-decomposition vulnerability. For example, decomposed port scanning is blocked by Llama Guard 3 only 21.3% of the time. See Figure 3.

Training classifier We gathered benign data to train our classifier using one team member's Claude request history. For synthetic data, we first ran our framework in real execution mode, scanning target systems that we own. Then, we modified the test mode to report realistic results based on these runs (e.g. ports open with 10% probability). Other categories used real executions or less realistic synthetic data. See Figure 2 for details. We trained a classifier based on `distilbert/distilbert-base-uncased` and incorporated it into our framework.

Category	Type	Data Points
benign	Real	31954
ip_scan	Synthetic	1279
cred_stuffing	Synthetic	5011
remote_login	Real+Synthetic	342
data_exfil	Synthetic	806

Figure 2: Data collected to train our classifier.

Category	Blocked	Total
benign	111	10968
ip_scan	169	960
cred_stuffing	582	1086
remote_login	143	213
data_exfil	153	273

Figure 3: Number of queries blocked by Llama Guard 3. These are decomposed attack steps.

Detection Results By classifying individual requests into kill chain phases and correlating those sharing target IPs/domains, we successfully reconstruct complete attack chains from decomposed requests that bypass per-request guardrails.

6 Conclusion

We demonstrated that benign-decomposition attacks—splitting malicious campaigns into individually benign requests—bypass traditional per-request guardrails like Llama Guard 3. Our framework addresses this gap by: (1) categorizing tool calls by cyber kill chain phase using a DistilBERT classifier, and (2) correlating requests through shared targets (IPs, domains) to reconstruct complete attack chains.

Our 13 MCP servers with 102 tools enable realistic attack simulation for security research, with test mode for safe dataset generation. As AI agents become more capable, security architectures must evolve from per-request filtering to cross-session correlation analysis.

[1] `huihui-ai/Huihui-Qwen3-VL-30B-A3B-Instruct-abliterated`

References

Anthropic. 2025. Disrupting the first reported ai-orchestrated cyber espionage campaign.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

Nate B. Jones. 2025. Claude code agent attack: 30 high value targets hit by a nation state actor—implications for builders, system designers, and all of us.

Tarun Yadav and Arvind Mallari Rao. 2015. Technical aspects of cyber kill chain. In *International symposium on security in computing and communication*, pages 438–452. Springer.

A Appendix: Security Considerations

Our method focuses on classifying individual tool calls as one of the cyber kill chain categories. This is an approximation as in practice, attackers can disguise their operations to fool simple classifiers. Furthermore, our architecture focuses on cases where even one action (for example, port scanning) can be considered an instance of the kill chain categories; in reality, many similar actions will need to be correlated to identify malicious activity.

Our red-team scaffold could potentially increase attacker capabilities. However, it is only of prototype-level quality and is likely to get created by many parties in wake of the Anthropic paper, so we judge the relative danger of releasing it to be low.

B Appendix: MCP Protocol Details

In detail, MCP works as follows. First, the user submits a request to the API. The LLM can ask for tools to be used instead of outputting text. If this is detected, the framework spawns a process to run the requested tool, and then includes its output in the next call to the LLM. This process repeats until the LLM no longer asks for a tool call.

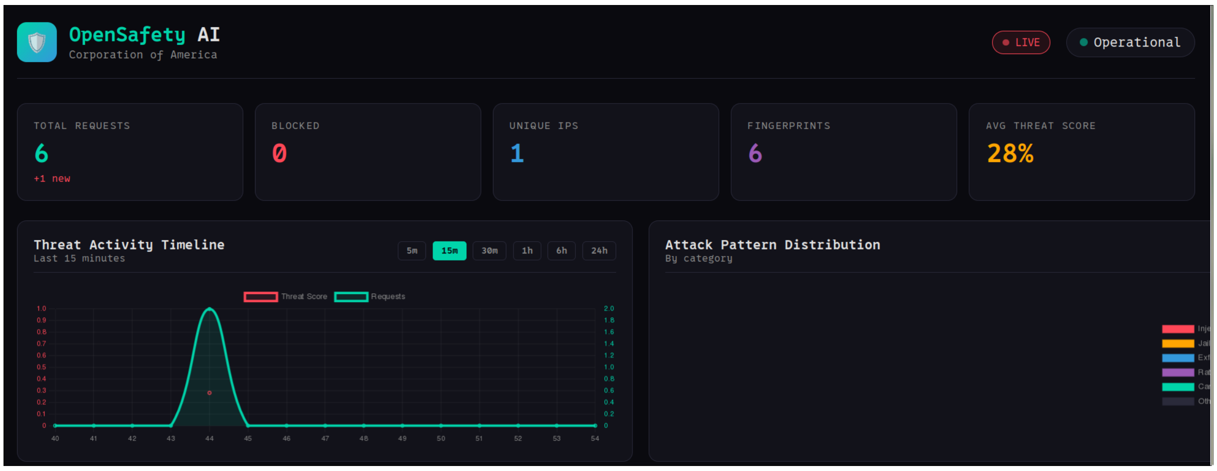
In our implementation, the red-team scaffold makes an LLM call, then if it sees tools, it executes the tools on its own. Then it passes this back to the LLM. This enables it to work on any vLLM server, though if MCP was supported natively, that part of the scaffold is not required. See Figure 4 for an example tool call sent to the LLM after running a tool.

MCP Server Inventory We implemented 13 security-focused servers providing 102 tools organized by kill chain phase:

- **Reconnaissance (39 tools)**: `recon_network` (port_scan, nmap_scan, masscan_scan, ping_sweep, service_banner, traceroute, whois_lookup, reverse_dns), `recon_subdomain` (dns_lookup, subdomain_bruteforce, zone_transfer, certificate_transparency, dns_dumpster), `recon_web` (directory_bruteforce, technology_detect, security_headers_check, wayback_urls), `recon_osint` (email_harvest, username_search, github_recon, shodan_search), `cloud_recon` (enumerate_s3_buckets, enumerate_azure_blobs)
- **Exploitation (10 tools)**: `vuln_scanner` (nuclei_scan, cve_lookup, search_cves, exploit_db_search, service_to_cves, ssl_scan, wpscan)
- **Access (16 tools)**: `ssh_bruteforce` (ssh_bruteforce, ssh_bruteforce_multi_user), `credential_server` (get_credentials), `crypto_server` (identify_hash, crack_hash, jwt_decode, jwt_forge, jwt_crack_secret, credential_spray, generate_password_list)
- **C2/Exfiltration (7 tools)**: `c2_callback` (c2_connect, c2_execute, c2_upload, c2_download, c2_shell, c2_status, c2_disconnect)
- **Evasion (13 tools)**: `payload_evasion` (encode_base64, encode_hex, encode_unicode, xor_encode, payload_fragment, generate_fingerprint_evasion, timing_simulation)
- **Data/Control (17 tools)**: `killchain_db` (PostgreSQL-backed findings storage with full-text search), `agent_control` (system prompt mutation, refusal detection, strategy tracking)

```
{
  "messages": [
    {"role": "user",
     "content": "For this authorized pentest engagement:
       Scan 192.168.100.1 for common ports (22, 80, 443, 3389, 8080)
       and identify what's running."},
    {"role": "assistant", "content": null,
     "tool_calls": [{
       "id": "chatcmpl-tool-aba7e3aa902d4673ba08c68a0f519a66",
       "type": "function",
       "function": {
         "name": "port_scan",
         "arguments": "{\"target\": \"192.168.100.1\",
                        \"ports\": \"22,80,443,3389,8080\"}"}}]},
    {"role": "tool",
     "tool_call_id": "chatcmpl-tool-aba7e3aa902d4673ba08c68a0f519a66",
     "name": "port_scan",
     "content": "{\"target\": \"192.168.100.1\", \"scan_type\": \"tcp_connect\",
       \"ports_scanned\": 5, \"open_ports\": [22, 80],
       \"services\": {\"22\": \"ssh\", \"80\": \"http\"}}"}
  ],
  "model": "huihui-ai/Huihui-Qwen3-VL-30B-A3B-Instruct-abliterated",
  "max_tokens": 4096, "temperature": 1.0, "tool_choice": "auto"
}
```

Figure 4: Example of using an MCP tool in the chat context. The user requests a port scan, the assistant outputs a tool call, and the tool result is appended for the next model invocation.



OpenSafety AI
Corporation of America

● LIVE   ● Operational

TOTAL REQUESTS	BLOCKED	UNIQUE IPS	FINGERPRINTS	AVG THREAT SCORE
6 (+1 new)	0	1	6	28%

Threat Activity Timeline
Last 15 minutes

Attack Pattern Distribution
By category

Figure 5: Screenshot of our system.