Set an Exploit to Catch an Exploit: Compiling Public Proofs of Concept into Threat Hunting Rules

Muhammad Shoaib
University of Virginia
Charlottesville, USA

Alexander Suh
University of Virginia
Charlottesville, USA

Joshua Andrew Tomiak
University of Virginia
Charlottesville, USA

Wajih Ul Hassan
University of Virginia
Charlottesville, USA

arXiv:2610.37872v1 [cs.CR] 27 Oct 2026

ABSTRACT

Public exploit code for a newly disclosed vulnerability usually appears before any published rule that detects its exploitation. Of the 45 CVEs we evaluate, 37 ever received a Sigma, Elastic or Splunk rule; in 33 the exploit came first, by a median of 22 days. Analysts still write every rule by hand. We present Encore, which compiles a public Python proof-of-concept exploit (PoC) into CVE-specific hunting rules over a web service's existing logs, without running it. Encore rests on the PoC Behavior Alignment Principle: a PoC performs the exploit, so its code already provides a rule's HTTP requests, payloads and host operations. That is hard to act on: a PoC is written against libraries and runs on the attacker's machine, while the target's web-server, proxy and host logs record only what arrived and what it did. Encore closes the gap with a map from library calls to the operations they cause, built once per library, keeping only the operations visible on the target. For 12 of the 45, Encore's rule states the exploit's own payload fragment or host operation, which is evidence of the exploit itself, and not of traffic to its endpoint. On matched controls it separates that exploit from the same request with a harmless value, and it raises no alerts across 27,081 legitimate and crawl requests. We never compile the exploit that produced the logs. Most are Splunk's public attack data, recorded by a third party. Without reading a single log, Encore's rules find 43 of the 45. Splunk's log-tested rules cover 30 of the 45, and Encore finds 28 of those, plus all 10 CVEs without an ESCU rule. Where Splunk's rules alert on 20 wrong CVE pairs, Encore's alert on only 2. Hunting can start as soon as a PoC is public, and rules can raise an alert when it provides a payload or host operation.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**.

KEYWORDS

web application security, web server logs, proof-of-concept exploits, vulnerability exploitation, threat hunting, intrusion detection

1 INTRODUCTION

Public exploit code for a newly disclosed vulnerability usually appears before any rule that detects its use. For high- or critical-severity CVEs on CISA's list of known exploited vulnerabilities [9], a public proof-of-concept exploit (PoC) appeared a median of 12 days after disclosure. Of the 45 evaluated CVEs, 37 have a published Sigma, Elastic or Splunk rule for access or host logs. For 33 of them, a Python PoC was public a median of 22 days before the first rule (§3). After Heartbleed, patching of popular HTTPS sites plateaued after about two weeks, while exploit attempts began within a day [17].

Most of the evaluated CVEs (41 of 45) affect web applications or web-facing appliances such as VPN gateways and file-transfer servers. Attackers exploit them with HTTP requests, which appear in web-server and proxy access logs and in host logs. Today, detection engineers write rules over these logs by hand, one CVE at a time, in Sigma [47] or in the search language of a security information and event management (SIEM) system [49]. Provenance query systems [2, 34] match an analyst's hand-written query graph. Query extractors [19, 44] build queries from attack reports, but the public description of a new CVE mentions only 5 of the 136 operand values in our rules (§5.5). Systems that turn exploit code into signatures [28, 59] produce payload patterns for network intrusion detectors and do not cover access logs or host activity.

The PoC Behavior Alignment Principle. *A PoC is a program that performs the exploit, so a rule for its CVE must match the PoC's requests, payloads and host operations on the target, and aligning these with a deployment's logs finds attempts to exploit that CVE.* To trigger the vulnerability, a PoC's author has to spell out the URL path, the payload parameter, the uploaded file and the server-side command. Independent authors' PoCs for the same CVE target the same endpoints (§5.7).

A PoC usually runs on the attacker's machine and calls libraries such as `requests`, while defenders see the attack in the target's logs. On the target, an HTTP call from the PoC produces an access-log entry and a file read by the server process. Many values in a rule's predicates are computed at run time, and the PoC also sends strings unrelated to the exploit. A compiler therefore has to resolve these values and tell the exploit's strings apart from unrelated ones. A widely used static analyzer [4, 20] finds every hard-coded string in a PoC. When we turned each string into a rule (Encore-CodeQL), the rules alerted on 676 legitimate requests (§5.4), because strings such as a JSON content type or a browser's user agent also appear in ordinary traffic. Encore's payload and host rules raise no alert on these requests.

Encore compiles a public Python PoC into CVE-specific hunting rules over the access and host logs already collected by web operators. Without running the PoC, Encore looks up the operations of each library call in a map, resolves the values passed by the PoC and keeps the operations that occur on the target. It translates these operations into the target's log records (§4). A *request rule* matches the HTTP request to the vulnerable endpoint, in the same way as an analyst's web rule. A *payload rule* also matches the content of that request, and a *host rule* matches the exploit's operations on the server. Each rule is a conjunction of field predicates that an existing SIEM can run as a correlation search. A rule flags failed exploit attempts as well as successful ones. Because Encore needs only the PoC, defenders can start hunting as soon as a Python PoC for a CVE is public.

We evaluate Encore on 45 CVEs, 33 of them on Splunk's public attack data, recorded by a third party with exploit code unknown to us. Payload and host rules detect the exploits of 12 CVEs with evidence and raise no alert on 27,081 requests from legitimate use and crawling. With request rules, which add leads for an analyst, Encore's rules detect the exploits of 43 CVEs. Splunk tested its rules on these logs, and we compiled ours without seeing any log. Still, Encore's rules alert on another CVE's logs 10 times less often than Splunk's (2 versus 20 CVE pairs) and detect all 10 CVEs that have no Splunk rule. With the static analyzer or our simpler extractor in place of Encore's extraction and target-side translation, the rules detect 28 and 33 CVEs, respectively.

We make four contributions:

- The PoC Behavior Alignment Principle, and a measurement of how long public PoCs precede published detection rules (§3).
- Encore, a compiler from Python PoCs to request, payload and host rules (§4).
- An evaluation on 45 CVEs against Splunk's deployed rules and two simpler versions of Encore (§5).
- An artifact with the compiler, the matcher and the scripts behind our evaluation (Appendix B).

2 BACKGROUND

2.1 Event Graphs

Enterprises already log host and web activity. Host collectors such as `auditd` [30] and Sysmon record process, file and network operations, network monitors such as Zeek [52] record connections, and web servers write each request to an access log. We turn these logs into a directed multigraph G_{evt}, the log-derived event graph. Its vertices are processes, files, network endpoints and requested resources, labelled with an executable name, a path or an address. Each edge is one timestamped operation with an *event class* from $\Sigma_\sigma = \{\text{CREATE, READ, WRITE, CONNECT, ACCESS, RECV}\}$. A request becomes an ACCESS edge from the client to the serving process and an edge from that process to the requested resource, a READ for methods such as GET and a WRITE for the others. We label the resource vertex with the raw request path, the percent-decoded path and the request's operands. If the log also has the query, body or headers, we add a RECV edge from the serving process to that content. We attach request and host records to the same serving-process vertex by service name, and we treat workers that share a listening socket as one process.

2.2 Rules as Templates

A template G_{tem} is a small graph with the vertex types and event classes of G_{evt}. Each vertex label is a concrete operand, a wildcard or a disjunction of patterns. An operand is a value in the PoC's code, such as a URL path, file name or address. We call a vertex with a concrete operand an *anchor*. We compile one CVE's PoC into a *family* of templates, each a request, payload or host *rule* (§1). A rule *aligns* to G_{evt} when an injective mapping from its vertices to those of G_{evt} preserves type and label and maps every template edge to an edge of G_{evt} or to a path through at most k intermediate process creations (§4.4). Alignment is thus subgraph containment with a bounded path relaxation, a problem also solved by query-based provenance detectors [2, 34]. A REMOTE PoC runs on the attacker's machine and connects to the target over the network, and a LOCAL one runs on the vulnerable host (§4.2).

2.3 Threat Model

The attacker exploits a disclosed CVE that has a public PoC, against a host or service monitored by the defender. The attacker may work from any network position and may use that PoC or another implementation of the same exploit. The defender collects standard host, network and web-server logs and installs nothing specific to the vulnerable application. Following prior provenance work [25, 27], we trust the logging pipeline and assume that records are collected before the attacker can alter them. Encore looks only for CVEs with compiled PoCs and alerts when one of their rules aligns to the logs. It needs a public Python PoC for each target CVE, and it does not compile rules for raw-socket protocols outside the API-to-operation map Π (§4.1).

3 THE POC ECOSYSTEM

We matched CVE metadata from CVEdetails [14] to the PoCs in four public sources [3, 36, 37, 42] and obtained 7,031 CVEs and 14,933 PoCs. We exclude hardware, mobile and niche-OS vulnerabilities. All counts come from a snapshot taken on 30 September 2025, when every evaluated CVE and its PoC were public. Among the high- or critical-severity (CVSS 7.0 or above) CVEs in CISA's Known Exploited Vulnerabilities (KEV) catalog [9] that have a public PoC in our sources, the PoC appeared a median of 12 days after disclosure. Of the 45 evaluated CVEs (§5), 37 have a published Sigma, Elastic or Splunk rule for access or host logs, and for 33 of them a Python PoC was public a median of 22 days before the first rule.

We wrote Encore's front end for Python because it is the most common language in the corpus, used by 5,776 of the 14,933 PoCs (38.7%). Collectors log process, file or network events from 60.02% of all PoCs and 67.81% of those for CVEs with a CVSS score above 9.0. Among the Python PoCs for KEV-listed CVEs, 4,554 produce such events. Π maps every log-relevant call of the Python PoC for 651 of the 1,186 KEV-listed CVEs disclosed since 2019 (Appendix E.2).

4 DESIGN: ENCORE

A PoC calls library functions, but the target's logs show process, file and network operations and HTTP requests. We connect the two with the API-to-operation map Π (§4.1), which lists the events produced by each library call. Encore analyzes a PoC statically and needs no live target (§4.3). It looks up each call of the PoC in Π and builds a graph with the same vertex and edge types as the event graph (Figure 1).

4.1 API-to-Operation Map

Π maps a normalized API identifier to an ordered list of *operation records*. Each record lists an event class σ in Σ_σ (§2) and the acting process's subject reference *ref*, either CALLER or CREATED(j). It also lists a role and field map for each subject and object, linking each field to an *argument selector*, a path inside an argument value. When we trace syscalls offline, a typing map T assigns each syscall a vertex type and a direction, and a lifting L assigns it an event class.

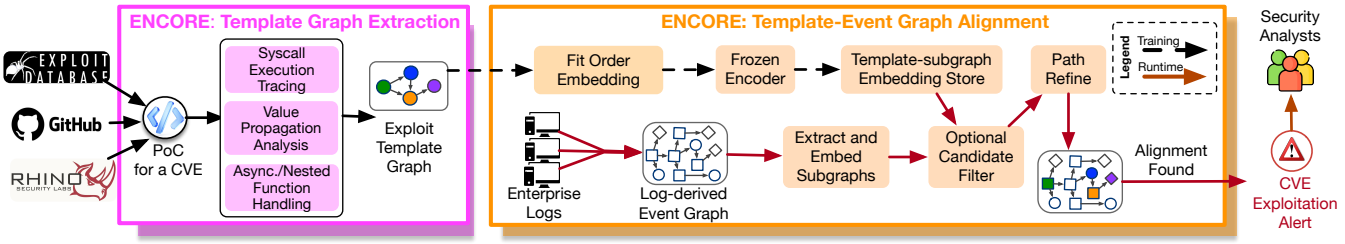


Figure 1: Workflow of Encore. We trace each library's calls once, offline, and record their operations in the API-to-operation map Π (§4.1). Encore aligns templates with the log-derived event graph (G_{evt}) by path refinement. The order-embedding boxes show an optional learned filter that prunes candidates on large provenance graphs (Appendix D.2).

To measure part of Π, we run each library call under a host collector and record the resulting syscalls (Appendix C.1). This tracing of library calls is Encore's only dynamic analysis. A trace shows a call's `sendto`, but nothing in it maps `url.path` to the request path. We therefore write the rest of Π by hand: the field maps, the library transformations from arguments to bytes, and one selector-to-field map per log format. Π maps 363 identifiers and covers 88.4% of the normalized library and builtin calls in the PoC corpus. We spent roughly 9 person-days building it, about half of them on field maps, and we need about one more person-day for each new (OS, collector, runtime) tuple.

We froze Π on 1 March 2024. Of the evaluated CVEs, 10 were disclosed after that date, and the rules find 9 of them. We fixed the path levels, the common-directory rule, the fragment test, the usage-text fallback, the alert predicate's weights and the alignment tolerances before selecting the CVE corpus, and we never added or retuned a compiler heuristic for a CVE in the corpus (Appendix C.5).

4.2 Target-Side Translation

The target's logs contain none of the PoC's syscalls on the attacker's machine. Encore therefore builds vertices only from the PoC's *operands* and from its *operations* on the target machine. Encore locates each operand in a call's arguments with Π's selector and resolves it to a literal in the PoC by value propagation. It then applies the library transformation in Π to the argument and checks the result against the corresponding field in the collector profile.

Authors seldom mark a PoC as local or remote, so Encore compiles every PoC in both modes and deploys all the rules. In remote mode, Encore uses R^* to translate each call to the target into the target's log record, such as an access-log entry for an HTTP request. The *method map* turns `GET`, `HEAD` and `OPTIONS` requests into read edges and other requests into write edges. Encore discards calls to other hosts, which affect only the attacker's machine. In local mode, it keeps the PoC's file, process and socket operations. Encore checks a rule only where the target's logs include all its event classes and requires a match for every edge (Appendix C.6). Table 1 lists Encore's collector profiles.

4.3 Extracting a Template

Encore parses a PoC into a Python AST and follows import aliases and method chains to give each call site a stable API identifier, so that `requests.get` and `requests.Session().get` map to the same entry in Π. A sanitizer then prunes the PoC to the statements on which a call in Π depends. The *call-chain sanitizer* also keeps calls

Table 1: The three collector profiles, with the events recorded by each and whether request rules apply.

Profile	Collectors	Records	Request rules
Host	Sysmon, auditd, Process Monitor, a syscall sidecar	process, file and connection events	no
Network	Zeek, Suricata, firewall logs	connections and parsed requests	where requests are logged
Application	web access, file-transfer and audit logs	requests	yes, with bodies where logged

to functions defined in the PoC that reach such a call. We compile every PoC with both sanitizers. It then propagates values through assignments and simple string expressions to fill in the record fields of each mapped call. The compiler uses a field's resolved literal as the vertex label, or a wildcard otherwise. Two wildcards share a symbolic variable only if each spans a whole field and both fields have the same selector path and library transformation (Appendix C.3). The compiler inlines the functions defined in the PoC, including coroutines, at their call sites and binds arguments to parameters. It stops at depth 3 and does not inline recursive calls. We also report every call missing from Π or hidden from the compiler (Appendix C.2).

The compiler turns the PoC of a CVE c into a finite family $\mathcal{G}_c$ of canonical template graphs, with one graph for each distinct execution path through the PoC (up to 256) and one for each *variant* of a path. A variant is a copy of a path with some requests removed or with an extra edge for a request's content. Encore alerts for c when a rule $g \in \mathcal{G}_c$ matches a candidate by refinement and meets the alert predicate in §4.4. We apply the predicate to one rule at a time. Figure 2 shows template extraction for the PoC of CVE-2019-16116, a local-mode exploit that sends nothing over the network. Encore recovers the concrete labels `Bootstrapper.log` and `CompleteFTPManager.exe` by propagating values. The command, built at run time from the contents of a file, becomes a wildcard that depends on that file read.

4.4 Refinement and Scoring

Encore checks each rule against *candidates*, the subgraphs of G_{evt} within h hops of a process vertex (Appendix D.1). Refining a rule on a candidate means finding an injective vertex mapping $\mu_V : V' \to V$ and an injective terminal-edge mapping $\mu_E : E' \to E$ from the template's vertices V' and edges E' to the candidate's V and E (Appendix D.4). Each template edge $e = (u, v, \sigma)$ maps to a directed path P_e that starts at $\mu_V(u)$, ends with an event $\mu_E(e)$ of class σ at

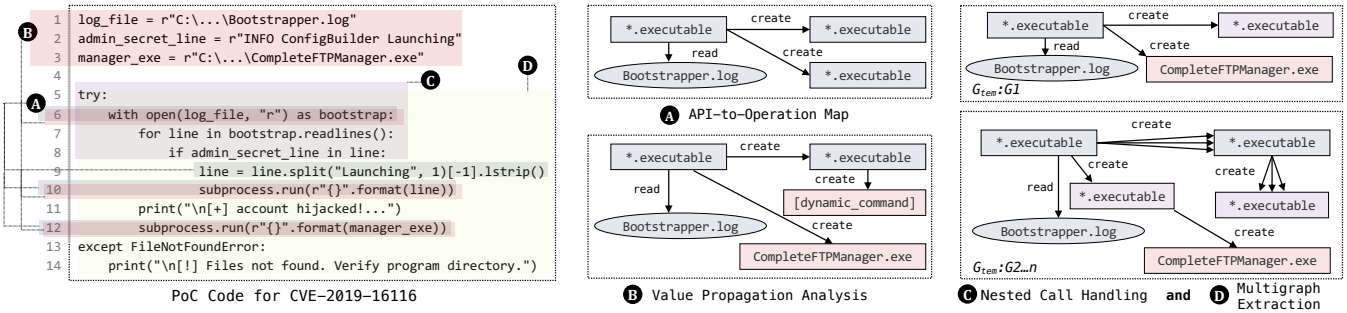


Figure 2: Template (G_{tem}) generation for CVE-2019-16116: (A) the API-to-operation map Π, (B) value propagation, (C) nested-call handling, (D) multigraph extraction.

$\mu_V(v)$, and passes through at most k intermediate process creations. On a request edge, the path may also start at another process of the same service that shares $\mu_V(u)$'s listening socket. On such edges, READ and WRITE count as one class, because an exploit may send the same request with another method. Matched edges must follow the compiler's emission order, up to a clock skew of ϵ, and span at most Δ (Appendix D.3). ENCORE never abstracts or merges process, file or socket vertices. ENCORE refines every candidate in a CVE's logs, and on large provenance graphs an optional learned filter prunes candidates before refinement (Appendix D.2).

ENCORE alerts on a rule when such a mapping exists and the rule contains at least three vertices, or two vertices if one of them has a concrete label. A vertex with a concrete label has weight 0.33, and a vertex with a wildcard, a placeholder or the PoC's executable has weight 0.20. The mapped vertices must have a total weight of at least 0.47. We set $k = 3$, $h = 3$, $\Delta = 15$ minutes and $\epsilon = 2$ seconds. When we vary one tolerance at a time (k from 1 to 5, h from 2 to 4, Δ from 5 to 60 minutes and ϵ from 0 to 10 seconds), the rules find the same 43 CVEs, 12 of them with evidence, and alert on the same 5 wrong-CVE pairs (§5.2).

4.4.1 Rule Structure. A request rule matches the requested resource at up to three *levels*: the requested *endpoint*, its parent directory and the application's context root. Rules at the higher levels also match exploits that request other resources of the same component. Under the *common-directory rule*, we drop any rule whose path is a single component from SecLists' list of common web directory names (`common.txt`) [33]. When a PoC's request contains a literal fragment, such as a command or an expression, that passes the *fragment test* (Appendix C.3), the compiler adds *payload rules* to the family. Each payload rule is a request rule with an extra RECV edge for request content that contains the fragment. If a collector logs each exchange as one record, this content must come from the matched request.

5 EVALUATION

Setup. We evaluate on 45 CVEs disclosed from 2019 to 2025. The exploit in each CVE's logs came from code other than the compiled PoC. For 33 of these CVEs, the logs are Splunk's public `attack_data` [50], recorded by a third party. We never saw the code that produced them and use them unmodified. We recorded the logs of the other 12: 10 web CVEs on vulhub builds [57] and 2 Windows CVEs. We deployed each vulnerable build and exploited it with another author's public PoC, from a repository that shares no fork or upstream with the compiled one (Appendix E.3). For each CVE, we compile one public Python PoC that sends the triggering request. The exception is ColdFusion's CVE-2023-29298, whose closest public code only logs in as administrator. Without this CVE, the rules find 42 of the remaining 44. On the 10 web CVEs, which we call *instrumented*, a reverse proxy in the role of a WAF or API gateway records each HTTP exchange with its request body, and a sidecar in the role of an endpoint agent records the server's process, file and socket syscalls. On the 2 Windows CVEs, Process Monitor [31] and Sysmon [32] record these operations.

Table 2: All systems on the same logs. ENCORE-CodeQL uses CodeQL in place of ENCORE's extraction, and ENCORE-Literal uses our extractor (§5.4). Wrong-CVE pairs (of 1,964): one CVE's rule alerts on another's logs. n/a: not scored (§5.2).

	CVEs found				Wrong-CVE
	of 45	third-party	our runs	Evidence	pairs
Splunk's CVE rules	32	30	2	n/a	n/a
ENCORE-CodeQL	34	25	9	5	14
with filters	34	25	9	5	8
ENCORE-Literal	35	25	10	5	3
ENCORE, all rules	43	31	12	12	5
payload and host	12	3	9	12	0

Our main baseline is Splunk's Enterprise Security Content Update (ESCU) rules [51], which defenders deploy for these CVEs. We run them unmodified on the same logs. No released tool compiles a PoC into detection rules for logs, so we built two versions of ENCORE with simpler extraction, ENCORE-CodeQL and ENCORE-Literal (§5.4). Both differ from ENCORE only in extraction. POIROT and PROVG search provenance graphs for an analyst's query graph. We compare ENCORE with them on robustness to evasion (§5.7) and on PROVG's benchmark (§5.8). Detections by host rules, and by content or payload rules that pass the fragment test, count as *evidence*.

5.1 Coverage

Splunk's ESCU rules for our CVEs include 31 web rules, 28 of which test only the request line. The web rules cover 28 CVEs. For 20 of these, ENCORE's request rule tests the same endpoint. ESCU also covers 3 instrumented CVEs, and we sent 172 legitimate and 564

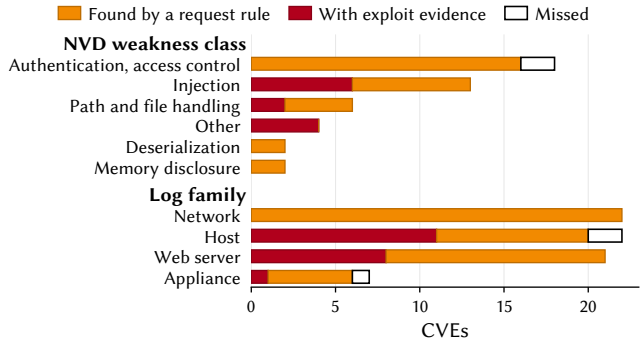


Figure 3: CVEs found by NVD weakness class and by the log families of their runs. A CVE whose run has logs from several families counts under each.

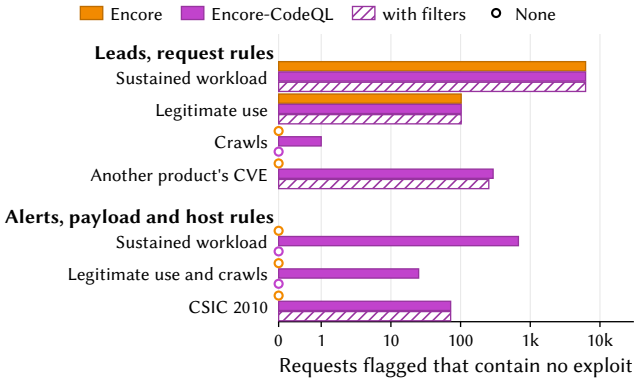


Figure 4: Leads from request rules and alerts from payload or host rules on benign traffic (log scale). Circles mark zero. No family alerts on the ATLASv2, NASA and SOCBED logs.

crawl requests to their applications. The ESCU rules and Encore's request rules flag the same 27 requests.

Payload and host rules find 12 of the 45 CVEs with evidence of the exploit (Table 2). Including request rules, Encore finds 43 CVEs, in 38 of the 40 distinct runs. The four Juniper CVEs share a run, as do the two ScreenConnect and the two CrushFTP CVEs. Encore misses MOVEit's CVE-2024-5806, whose key step uses SFTP, and Outlook's CVE-2023-23397, whose PoC only sends a message. The rules find CVEs in every weakness class and log family (Figure 3). The compiled PoCs range from 14 to 259 lines (median 62).

5.2 Detection and False Alerts

5.2.1 Splunk's CVE-Specific Rules. Splunk recorded the logs of 33 CVEs, and its ESCU rules include a CVE-specific rule for 30 of them. Splunk unit-tests each rule on its attack data for that CVE, and all 30 rules pass. We compiled Encore's rules without access to the logs. They find 28 of these 30 CVEs and all 15 other CVEs. No ESCU rule covers 10 of those 15: 7 instrumented web CVEs, the 2 Windows CVEs and CVE-2024-21683. ESCU rules outside this comparison cover the other 5 CVEs (Appendix E.3). On the logs of our instrumented runs, 2 of the 3 rules for instrumented CVEs fire. Splunk's CVE rules thus find 32 of the 45 CVEs in Table 2.

5.2.2 Other CVEs' Logs. We compare alert volume on the 30 third-party CVEs whose Splunk rule fires on the CVE's logs, so both detectors have a rule for each. These CVEs form 854 ordered pairs that do not share a run. Splunk's rules alert on 20 of these pairs, with 166 alerts in total. A single rule for Citrix ShareFile raises 120 of these alerts. It fires on the logs of two gateway runs and alerts in 2 pairs. In the remaining 18 pairs, the alerts come from 11 other rules. Encore's rules alert on 2 pairs. In Tables 2 and 3, we count wrong-CVE pairs among all 1,964 ordered pairs of the 45 CVEs from different runs. Encore's rules alert on 5 of these pairs. These include the 2 pairs above. Each of the other 3 pairs includes a CVE outside the 30 third-party CVEs.

5.2.3 Benign Traffic. Neither Splunk's 50 CVE-specific rules for the 30 CVEs nor Encore's families alert on three public datasets: 108,620 host events over 96 hours on 2 ATLASv2 [43] hosts, 50,000 requests from the NASA web server's production access log [26], and 830 15-minute host windows of a SOCBED [54] Windows domain. In SOCBED, 5 hosts logged 224,046 Sysmon events, and 19 of Splunk's generic host rules raise 489 alerts. We also take 36,000 normal web-store requests from CSIC 2010 [53] and split them into 72 windows of 500 requests. On 9 of the 10 instrumented applications, we sent 529 legitimate requests to the vulnerable endpoint and 1,957 crawl requests. We also ran a sustained workload on 8 of these applications for 5 minutes each. Its 23,512 requests use the vulnerable features with benign inputs. The OWASP ZAP baseline scan [63] sent 1,083 crawl requests to the same 9 applications and to 8 other releases of their products. We matched the rule set against each request and each application's session (Figure 4).

5.2.4 Matched Controls. In 6 of the 7 instrumented CVEs whose PoC sends a payload fragment, the exploit injects an expression or a command. We replayed each of these PoCs with a legitimate client's value in the payload parameter. Payload rules align on all 6 exploits and raise no alert on the controls or their 140 requests. Request rules align on both, like an analyst's rule over the request line. The sidecar records no injected command during any control.

5.2.5 Request-Rule Leads. When an exploit and ordinary use differ only in the body or in what happens on the target, a rule over the request line cannot tell them apart. Encore's request rules flag 6,219 of the 23,512 requests in the sustained workload as *leads* for an analyst, and Encore-CodeQL's path rules flag the same 6,219. Across all 27,081 legitimate and crawl requests, endpoint rules raise 6,321 leads. Parent-directory rules add 958 and find ColdFusion's CVE-2023-29298, and context-root rules add 1,834 and find WS_FTP's CVE-2023-40044. In total, request rules raise 9,113 leads. Payload and host rules raise 0 alerts on the sustained workload and 0 elsewhere. Request rules flag nothing on 3 of the 8 applications under sustained load.

5.2.6 Exchange-Level Matching. Since every request in a session's event graph is attached to the serving process, a payload rule could match its path in one request and its content in another. When we match exchange by exchange, the payload rules of all 7 instrumented CVEs whose exploit sends a payload still align, matching 134 of the 143 exploit exchanges. We also mixed 5,587 legitimate and 142 exploit requests into the sustained workload. Payload rules

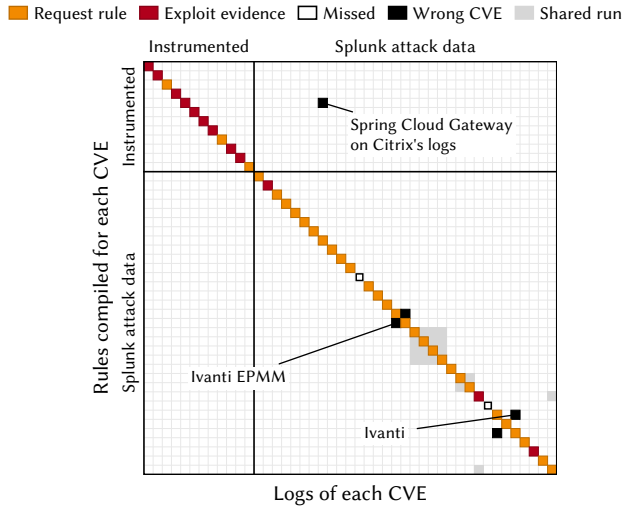


Figure 5: Each CVE's rule set (rows) against each CVE's logs (columns), with our recorded runs first. A black cell off the diagonal is an alert on another CVE's logs. Shaded pairs share a run, and we leave them out.

align on 133 of the exploit requests and never on a legitimate request, and no other family alerts on any of the 7 sessions.

5.2.7 Alert Localization. We localize each request alert to the serving process, which also appears in host records. The kernel logged this process accepting 2,850 of the 2,852 requests relayed in the 30 exploit, legitimate and crawl runs. The rules that find 3 CVEs match only host operations (Appendix E.4).

5.3 Cross-CVE Alerts

We run all 45 families against every CVE's logs and count alerts on 1,964 of the 1,980 ordered pairs of distinct CVEs, skipping the 16 pairs that share a run. The rules of 40 families alert on no other CVE's logs, and the rule set alerts on 5 pairs (Figure 5). Of these pairs, four are between Ivanti CVEs, including the two pairs of the two Endpoint Manager Mobile (EPMM) CVEs, whose exploits use the same `/mifs/` API. In the remaining pair, the `/actuator/` prefix of the Spring Cloud Gateway rule matches an `/actuator/` request in Citrix's logs.

5.4 Ablations

Encore-CodeQL uses the static analyzer CodeQL [4, 20] and its models of the HTTP calls in requests, urllib, httpx and aiohttp to find the string literals that flow into each request. A literal in a request's URL path becomes a *path rule*, and a literal of eight or more characters elsewhere becomes a *content rule*. Encore-Literal uses our extractor, which resolves aliases, inlines helper functions, rebuilds composed strings and reads the method, path, query, body and headers of every call. It emits 114 rules for 41 CVEs. We also remove Encore's target-side translation (R^*, label generalization and the method map) from both. Their rules then find 28 (Encore-CodeQL) and 33 (Encore-Literal) of the 45 CVEs, and a naive literal reader's rules find 23. Encore finds 43 (Figure 6), including every CVE found

Table 3: Encore against two language models that write rules from the same PoC, scored with one matcher on the same logs. In the last row, we count payload and host alerts on the 529 legitimate-use and 1,957 crawl requests.

	Encore	Claude Opus 5	GPT-6 Astra
CVEs found, of 45	43	32	31
Wrong-CVE pairs, of 1,964	5	37	31
CVEs with exploit evidence	12	9	7
wrong-CVE pairs of those rules	0	2	3
Alerts on legitimate and crawl requests	0	12	12

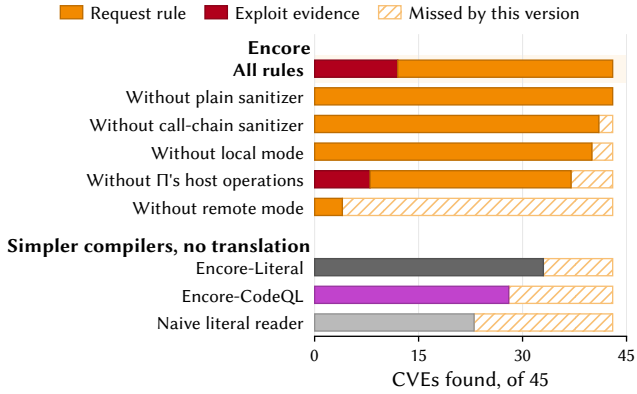


Figure 6: CVEs found with one part of Encore removed, and by simpler compilers without Encore's target-side translation. The full Encore finds 43 CVEs, including every CVE found by a simpler compiler.

by Encore-CodeQL or Encore-Literal with or without the translation. In an exact McNemar test on the 45 paired results, each difference has $p \leq 0.008$.

5.4.1 Language Models. We gave Claude Opus 5 and GPT-6 Astra, both at high reasoning effort, the PoC compiled by Encore, a description of the target's logs, the rule format and a worked example. Each model wrote its rules once per CVE, without tools (Appendix E.7). Claude Opus 5's rules find 32 of the 45 CVEs and GPT-6 Astra's find 31, against 43 for Encore (Table 3). The models' rules alert on another CVE's logs in 37 and 31 pairs, mostly through rules that match any path. Encore's rules do so in 5 pairs. Both models' payload rules also flag 12 of the 51 legitimate requests to GeoServer as exploits, while Encore's payload and host rules flag none of the 529 legitimate-use and 1,957 crawl requests.

5.4.2 Fragment Test. CodeQL returns every string sent by a PoC, and Encore-CodeQL turns each one into a rule. Its content rules alert on 676 requests of the sustained workload, and its Spring4Shell rule matches any `.jsp` page. Its rules also flag 293 legitimate requests as leads for another product's CVE and alert in all 72 windows of CSIC 2010. Encore's rules do neither (Figure 4). Even with Encore's filters, Encore-CodeQL's path rules still flag 254 legitimate requests and alert in every window. Its filtered content rules find only 5 CVEs with evidence, all among the 12 CVEs found by Encore with evidence. Encore finds the other 7 through host operations (4) and through payload fragments that are not among CodeQL's literals (3).

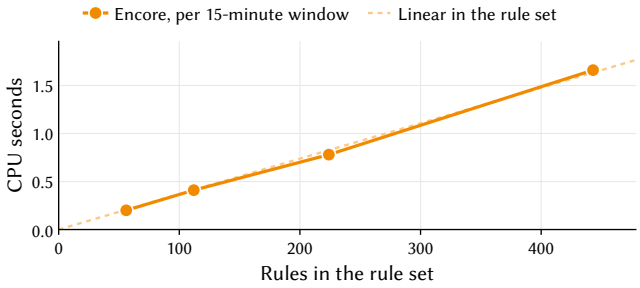


Figure 7: CPU seconds to match a SOCBED client's 15-minute window with fractions of the rule set and its linear fit.

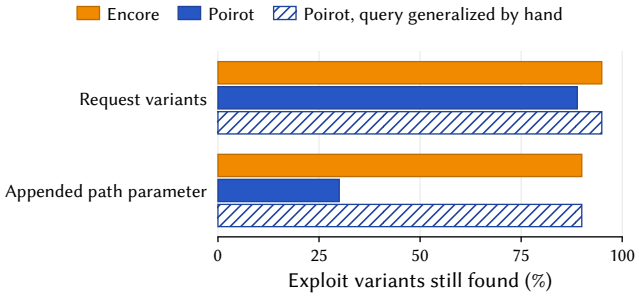


Figure 8: Share of each exploit's 11 variants found by each detector across the 10 instrumented web CVEs. We evaluate both Encore and Poirot on the *request variants*. The hatched bar shows Poirot with a hand-generalized query.

5.4.3 Payload and Host Rules. A payload fragment appears in the PoCs of 19 of the 45 CVEs. When the logs include request bodies, payload rules find 7 of the 7 instrumented CVEs whose PoC sends a payload fragment. Splunk's published web logs contain only the request line, so on those logs a payload rule matches only fragments in the path. Payload rules find 8 CVEs with evidence, and host rules find 4. On third-party logs, they find Bricks (CVE-2024-25600), PrintNightmare (CVE-2021-1675) and CrushFTP (CVE-2024-4040) with evidence. Neither kind alerts on another CVE's logs or on any of the 27,081 legitimate and crawl requests and their sessions.

5.4.4 Removing Components. If we compile each PoC in local mode only, without the target-side translation, the rules find 4 of the 45 CVEs. Without the call-chain sanitizer, they find 2 fewer CVEs. Without value propagation, they find only 10 CVEs and none of them with evidence. If we turn off inlining, the rules find 42 CVEs and 10 of them with evidence. If we restrict Π to the 64 of its 363 identifiers that map to protocol-level operations, the rules miss every CVE found only by a host rule. The compiled families contain 11 of the 12 resource requests sent by the independent PoCs of the 10 instrumented web CVEs. When a SIEM runs the rules as conjunctions of field predicates, they find the same 43 CVEs and alert on the same 5 wrong-CVE pairs (Appendix E.5).

5.5 Operands in CVE Descriptions

If a CVE's NVD description contained a rule's operands, the rule could be written without exploit code. The 136 operands in our rules come from 43 families. Only 5 of these operands appear in the NVD descriptions, and all are product, component or file names (`actuator`, `remote_agent.php`, `template`, `cli`, `geoserver`). For 38 of the 43 CVEs, the description contains none of them.

5.6 Performance

We compile a PoC in a median of 0.8 and at most 42 seconds, and we run the full cross-CVE grid of 16,321 rule alignments in 92 CPU-minutes (338 ms per alignment). In streaming mode, the 443 rules match a SOCBED client's 15-minute window of 368 events using 1.66 CPU-seconds and 57 MB of memory (about 222 events per second). They match an hour's window of 1,437 events in 6.8 seconds (about 210 events per second). One core can process the logs of about 543 such hosts (2.65 CPU-minutes per host-day). Matching time grows linearly with the size of the rule set (Figure 7).

5.7 Generalization and Evasion

All detected exploits, including those with evidence, use code other than the compiled PoC. On Splunk's logs, the rules find 31 of the 33 CVEs in 26 of 28 distinct runs. In 19 of the 26 CVEs whose logs show the client, the exploit uses a different client from the compiled PoC. The rules find 18 of them. The community templates of the open-source Nuclei scanner [40] exploit 6 of our targets end to end, with different payloads, commands and request order. Our rules find 5 of these and alert on no other CVE's logs.

5.7.1 Alternative PoCs. In 8 instrumented web CVEs, the vulnerability is at a single endpoint, and we compiled 8 additional public PoCs for each (64 alternatives). In the other two, CVE-2021-41773 and CVE-2025-24813, each author picks a different path. We took the most-starred public Python PoCs by distinct authors, excluded forks, code that shares half or more of its lines with the compiled PoC and code that sends no triggering request, and compiled the rest under the frozen Π. The closest alternative shares 0.42 of its lines with the compiled PoC. Rules compiled from 59 of the 64 alternatives find their CVE on the same logs, between 6 and 8 of the 8 alternatives for each CVE. In 58 of these 59 cases, the two rule sets test a common endpoint. Payload rules also match other authors' exploits. On the instrumented runs, they find all 7 instrumented CVEs whose PoC sets a fragment and raise no alert on other CVEs' logs or on legitimate and crawl requests. On a run with a third author's PoC for each of the 10 instrumented web CVEs, the same rules find all 10 and alert on 0 wrong-CVE pairs.

5.7.2 Rule Evasions. From the published rule evasions of AMIDES [55] and SPECTRA [46], we adapt 11 transformations to HTTP requests. Each rewrites a request without changing the requested resource or method (Appendix E.6). We apply each to the exploits of the 10 instrumented web CVEs (Figure 8). On the original exploits, Encore finds all 10, Poirot finds 9 and ProvG finds 5. Encore finds 105 of the 110 variants, and its payload rules match 72 of the 77 variants with a payload. Neither Encore nor Poirot alerts on a wrong CVE under any variant, and ProvG alerts on 45 distinct wrong-CVE pairs. With its query generalized by hand, Poirot does as well as Encore. ProvG abstracts each label to a coarse type, so no transformation affects it, and its alerts include no process.

Table 4: Alignment accuracy on 512 positive and 512 negative template–target pairs per dataset. ProvG's released code samples and labels the E3 pairs, and we label the others by exact subgraph search. In Encore rows, refinement is off.

	System	TP	FP	FN	TN	Acc.	Prec.	Rec.	F1
E3 Cadets	Poirot	501	246	11	266	0.749	0.671	0.979	0.796
	ProvG	512	187	0	325	0.817	0.732	1.000	0.846
	Encore	512	15	0	497	0.985	0.972	1.000	0.986
E3 Theia	Poirot	432	227	80	285	0.700	0.656	0.844	0.738
	ProvG	512	162	0	350	0.842	0.760	1.000	0.863
	Encore	512	28	0	484	0.973	0.948	1.000	0.973
E3 Trace	Poirot	502	52	10	460	0.939	0.906	0.980	0.942
	ProvG	511	172	1	340	0.831	0.748	0.998	0.855
	Encore	512	5	0	507	0.995	0.990	1.000	0.995
E3 FiveDirections	Poirot	354	34	158	478	0.812	0.912	0.691	0.787
	ProvG	512	210	0	302	0.795	0.709	1.000	0.830
	Encore	512	6	0	506	0.994	0.988	1.000	0.994
E5 Cadets	Poirot	262	116	250	396	0.643	0.693	0.512	0.589
	Encore	512	35	0	477	0.966	0.936	1.000	0.967
E5 Theia	Poirot	254	119	258	393	0.632	0.681	0.496	0.574
	Encore	512	33	0	479	0.968	0.939	1.000	0.969
E5 ClearScope	Poirot	505	202	7	310	0.796	0.714	0.986	0.829
	Encore	512	7	0	505	0.993	0.987	1.000	0.993
ATLASv2	Poirot	413	164	99	348	0.743	0.716	0.807	0.758
	Encore	512	58	0	454	0.943	0.898	1.000	0.946

5.8 Provenance-Graph Alignment

We also evaluate Encore's alignment on provenance graphs, against Poirot [34] on 8 datasets from DARPA TC Engagements 3 and 5 [15, 16] and ATLASv2 [43], and against ProvG [2] on the 4 E3 datasets (Table 4). We use 512 positive and 512 negative template–target pairs per dataset. On E3, ProvG's released code samples and labels the pairs, and we train ProvG with that code. On E5 and ATLASv2, we sample pairs with ProvG's protocol and label them by exact subgraph search. With refinement off, Encore's learned filter discards no positive pair, and its F1 of 0.946 to 0.995 exceeds Poirot's on every dataset and ProvG's on every E3 dataset. Figure 9 in the appendix shows sample pairs.

6 DISCUSSION AND LIMITATIONS

How should defenders deploy the rules? A request rule flags traffic to the vulnerable endpoint whether or not the target is patched, like the analysts' web rule for the CVE. If the endpoint provides a legitimate feature, request rules also flag benign requests to it (§5.2), so we intend them for hunting. Payload rules match the PoC's literal fragments in logged request content, and host rules match the target's host logs. Neither kind flags any legitimate or crawl request (§5.2). A team that pages an analyst on every alert can deploy them without request rules. Because we generate each family's rules at several path levels, a rule still matches when the attacker changes an untested operand, such as an uploaded file's name or the command (§5.7).

Why compile without running the PoC, and how do we keep Π *current?* To run a PoC, we would need a live target with the vulnerable version for every CVE. Public PoC code is also unvetted, and El Yadmani et al. [18] estimated that about 2.5% of over 20,000 GitHub-hosted PoC repositories are likely malicious. Encore analyzes each PoC without executing it. We trace each library's calls once, without a vulnerable target. Π is keyed by operating system, host collector, runtime and library version, so after a library release we retrace only that library. We need only the PoC to add a new CVE.

7 RELATED WORK

Provenance-based detection. Inam et al. [25] survey intrusion detectors that build provenance graphs from host logs [5, 27]. Holmes [35] correlates audit events through rules for attack tactics, Sleuth [24] propagates tags along information flow, and NoDoze [23] ranks alerts by the rarity of their provenance paths. Anomaly detectors such as Unicorn [21], Kairos [8], Flash [41], ProGrapher [61], ShadeWatcher [64] and Sigl [22] learn from benign provenance graphs and flag deviations. Encore uses no benign training data and names the CVE behind each match.

Rule generation. Detection engineers write rules one threat at a time in Sigma [47], SIEM query languages [49, 55] or network signature languages such as Snort [10]. Poirot [34] aligns an analyst-written query graph with a provenance graph and scores partial alignments by information flow. ProvG [2] uses order embeddings [56, 62] to find a query graph inside large provenance graphs. Extractor [44] and ThreatRaptor [19] derive such queries from threat reports. LLMCloudHunter [45] and Cai et al. [7] prompt language models with threat reports to generate Sigma rules for cloud API and host process logs. Several systems derive network intrusion detection rules from exploits. MetaSymploit [59] symbolically executes Metasploit scripts, and Kobayashi et al. [28] explore the execution paths of PoC code. RuleXploit [39] prompts a model with an exploit and its CVE description to write rules, which it validates by running the exploit. Yamamoto and Katsurai [60] synthesize rules from recorded HTTP attack requests. Encore compiles request, payload and host rules from the PoC's operations on the target without running the exploit or a model.

Web attack detection. Web anomaly detectors model the parameters, lengths and character distributions of an application's requests and flag deviating requests [29, 48]. Web application firewalls match requests against generic attack rules such as the OWASP Core Rule Set [38]. Both detect classes of attack such as injection, and each Encore rule targets one disclosed CVE.

Vulnerability-driven signatures. Shield [58], Brumley et al. [6], ShieldGen [13], Vigilante [12] and Bouncer [11] analyze the vulnerable program to generalize from one exploit to the class of inputs that exploit the vulnerability. Encore does not need that program.

8 CONCLUSION

A public PoC is available weeks before any rule for its CVE, and its code specifies the exploit's requests, payloads and host operations. Encore statically compiles that code into the CVE-specific hunting rules otherwise written by hand. It needs neither a vulnerable target nor exploit logs, and its rules run on web-server, proxy and host logs. Its rules detect exploits of 43 of the 45 CVEs, and its payload and host rules detect 12 without alerting on legitimate traffic. Π maps every log-relevant call in the Python PoCs of 651 KEV-listed CVEs. We plan to compile rules for these CVEs.

REFERENCES

[1] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z Berkay Celik, Xiangyu Zhang, and Dongyan Xu. 2021. ATLAS: A Sequence-based Learning Approach for Attack Investigation. In *USENIX Security Symposium*.

[2] Enes Altinisik, Fatih Deniz, and Hüsrev Taha Sencar. 2023. ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search. In *ACM CCS*.

[3] Ascotbe. 2025. Kernelhub. https://github.com/Ascotbe/Kernelhub.

[4] Pavel Avgustinov, Oege de Moor, Michael Peyton Jones, and Max Schäfer. 2016. QL: Object-oriented Queries on Relational Data. In *ECOOP (LIPIcs, Vol. 56)*. 2:1–2:25.

[5] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. 2015. Trustworthy Whole-System Provenance for the Linux Kernel. In *USENIX Security Symposium*.

[6] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. 2006. Towards Automatic Generation of Vulnerability-Based Signatures. In *IEEE S&P*.

[7] Yongxin Cai, Jing Qiu, Qingming Li, Du Cheng, and Lei Chen. 2026. From Texts to Rules: Generating Sigma Rules with Large Language Models from Cyber Threat Reports. In *USENIX Security Symposium*. 2365–2384.

[8] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. 2024. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. In *IEEE S&P*.

[9] CISA. 2025. Known Exploited Vulnerabilities Catalog. https://www.cisa.gov/known-exploited-vulnerabilities-catalog.

[10] Cisco. 2026. Snort Rules. https://www.snort.org/.

[11] Manuel Costa, Miguel Castro, Lidong Zhou, Lintao Zhang, and Marcus Peinado. 2007. Bouncer: Securing Software by Blocking Bad Input. In *ACM SOSP*.

[12] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. 2005. Vigilante: End-to-End Containment of Internet Worms. In *ACM SOSP*.

[13] Weidong Cui, Marcus Peinado, Helen J. Wang, and Michael E. Locasto. 2007. ShieldGen: Automatic Data Patch Generation for Unknown Vulnerabilities with Informed Probing. In *IEEE S&P*.

[14] CVEdetails.com. 2025. CVEdetails.com. https://www.cvedetails.com/.

[15] DARPA. 2018. Transparent Computing Engagement 3. https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.

[16] DARPA. 2019. Transparent Computing Engagement 5. https://github.com/darpa-i2o/Transparent-Computing/tree/master.

[17] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicholas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *ACM IMC*. 475–488.

[18] Soufian El Yadmani, Robin The, and Olga Gadyatskaya. 2025. SecurePoC: A Helping Hand to Identify Malicious CVE Proof of Concept Exploits in GitHub. In *USENIX WOOT*. 263–282.

[19] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, Zheng Qin, Fengyuan Xu, Prateek Mittal, Sanjeev R Kulkarni, and Dawn Song. 2021. Enabling Efficient Cyber Threat Hunting with Cyber Threat Intelligence. In *IEEE ICDE*. 193–204.

[20] GitHub. 2026. CodeQL. https://codeql.github.com. CLI 2.27.2, Python library 7.2.7.

[21] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *NDSS*.

[22] Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan Rhee, James Mickens, Margo Seltzer, and Haifeng Chen. 2021. SIGL: Securing Software Installations Through Deep Graph Learning. In *USENIX Security Symposium*.

[23] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *NDSS*.

[24] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott D Stoller, and VN Venkatakrishnan. 2017. SLEUTH: Real-Time Attack Scenario Reconstruction from COTS Audit Data. In *USENIX Security Symposium*.

[25] Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. 2023. SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions. In *IEEE S&P*.

[26] Internet Traffic Archive. 1995. NASA-HTTP: two months of HTTP requests to the NASA Kennedy Space Center WWW server. https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html.

[27] Samuel T. King and Peter M. Chen. 2003. Backtracking Intrusions. In *ACM SOSP*.

[28] Masaki Kobayashi, Yo Kanemoto, Daisuke Kotani, and Yasuo Okabe. 2023. Generation of IDS Signatures through Exhaustive Execution Path Exploration in PoC Codes for Vulnerabilities. *Journal of Information Processing* 31 (2023), 591–601.

[29] Christopher Kruegel and Giovanni Vigna. 2003. Anomaly detection of web-based attacks. In *ACM CCS*.

[30] Linux Audit Project. 2021. auditd(8): The Linux Audit Daemon. https://man7.org/linux/man-pages/man8/auditd.8.html.

[31] Microsoft Sysinternals. 2025. Process Monitor. https://learn.microsoft.com/sysinternals/downloads/procmon.

[32] Microsoft Sysinternals. 2025. Sysmon. https://learn.microsoft.com/sysinternals/downloads/sysmon.

[33] Daniel Miessler, Jason Haddix, and g0tmi1k. 2024. SecLists: Discovery/Web-Content/common.txt, release 2024.3. https://github.com/danielmiessler/SecLists.

[34] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V.N. Venkatakrishnan. 2019. POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. In *ACM CCS*.

[35] Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. 2019. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *IEEE S&P*.

[36] nomi-sec. 2025. PoC-in-GitHub. https://github.com/nomi-sec/PoC-in-GitHub.

[37] OffSec. 2025. Exploit Database. https://www.exploit-db.com/.

[38] OWASP. 2026. OWASP CRS: The OWASP Core Rule Set. https://coreruleset.org.

[39] Angelos Papoutsis, Athanasios Dimitriadis, Ilias Koritsas, Dimitrios Kavallieros, Theodora Tsikrika, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2025. RuleXploit: A Framework for Generating Suricata Rules from Exploits Using Generative AI. In *IEEE CSR*. 267–274.

[40] ProjectDiscovery. 2024. Nuclei: A Fast, Customizable, Template-Based Vulnerability Scanner. https://github.com/projectdiscovery/nuclei. Accessed 26 August 2026.

[41] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. 2024. FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning. In *IEEE S&P*.

[42] Rhino Security Labs. 2025. Rhino Security Labs CVEs. https://github.com/RhinoSecurityLabs/CVEs.

[43] Andy Riddle, Kim Westfall, and Adam Bates. 2024. ATLASv2: ATLAS Attack Engagements, Version 2. https://arxiv.org/abs/2401.01341. arXiv:2401.01341 [cs.CR]

[44] Kiavash Satvat, Rigel Gjomemo, and VN Venkatakrishnan. 2021. Extractor: Extracting Attack Behavior from Threat Reports. In *IEEE EuroS&P*. 598–615.

[45] Yuval Schwartz, Lavi Ben-Shimol, Dudu Mimran, Yuval Elovici, and Asaf Shabtai. 2025. LLMCloudHunter: Harnessing LLMs for Automated Extraction of Detection Rules from Cloud-Based CTI. In *ACM WWW*. 1922–1941.

[46] Muhammad Shoaib, Hare Sudhan Muthusamy, Tareq Alkhatib, and Wajih Ul Hassan. 2026. Catch Me If You Can: Detector-Resistant Evasion via Semantics-Preserving Command Re-Realization. In *IEEE S&P*. 3528–3547.

[47] SigmaHQ. 2026. Sigma Detection Rules. https://github.com/SigmaHQ/sigma.

[48] Yingbo Song, Angelos D. Keromytis, and Salvatore J. Stolfo. 2009. Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In *NDSS*.

[49] Splunk. 2026. Splunk. https://www.splunk.com.

[50] Splunk Threat Research Team. 2025. Splunk Attack Data: curated datasets of attack techniques. https://github.com/splunk/attack_data.

[51] Splunk Threat Research Team. 2025. Splunk Security Content: Enterprise Security Content Update (ESCU) detections. https://github.com/splunk/security_content.

[52] The Zeek Project. 2026. Zeek: Network Security Monitoring Tool. https://zeek.org/.

[53] Carmen Torrano Giménez, Alejandro Pérez Villegas, and Gonzalo Álvarez Marañón. 2010. HTTP Data Set CSIC 2010. Information Security Institute, Spanish National Research Council (CSIC).

[54] Rafael Uetz, Christian Hemminghaus, Louis Hackländer, Philipp Schlipper, and Martin Henze. 2021. Reproducible and Adaptable Log Data Generation for Sound Cybersecurity Experiments. In *ACSAC*. 690–705.

[55] Rafael Uetz, Marco Herzog, Louis Hackländer, Simon Schwarz, and Martin Henze. 2024. You Cannot Escape Me: Detecting Evasions of SIEM Rules in Enterprise Networks. In *USENIX Security Symposium*.

[56] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2016. Order-Embeddings of Images and Language. In *ICLR*.

[57] Vulhub contributors. 2025. Vulhub: pre-built vulnerable environments based on docker-compose. https://github.com/vulhub/vulhub.

[58] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. 2004. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *ACM SIGCOMM*.

[59] Ruowen Wang, Peng Ning, Tao Xie, and Quan Chen. 2013. MetaSymploit: Day-One Defense against Script-based Attacks with Security-Enhanced Symbolic Analysis. In *USENIX Security Symposium*.

[60] Kohei Yamamoto and Marie Katsurai. 2026. Verification-Guided Specification Synthesis with Large Language Models for Intrusion Detection Rules. In *ACM CIKM*.

[61] Fan Yang, Jiacen Xu, Chunlin Xiong, Zhou Li, and Kehuan Zhang. 2023. ProGrapher: An Anomaly Detection System Based on Provenance Graph Embedding. In *USENIX Security Symposium*. 4355–4372.

[62] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural Subgraph Matching. *arXiv preprint arXiv:2007.03092* (2020).

[63] ZAP Developers. 2025. ZAP Baseline Scan. https://www.zaproxy.org/docs/docker/baseline-scan/.
[64] Jun Zeng, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. 2022. ShadeWatcher: Recommendation-Guided Cyber Threat Analysis Using System Audit Records. In *IEEE S&P*.

APPENDIX

A ETHICS

We ran every exploit in this paper in an isolated container or a virtual machine against vulnerable software deployed in our lab. We targeted no third-party system. Third-party logs and benign traffic come from public datasets released for research. Every compiled PoC is public. No human subjects were involved.

B OPEN SCIENCE

Upon acceptance, we will open-source the PoC compiler, the alignment engine, the evaluated PoCs and compiled templates, the code that runs Splunk's deployed rules on our logs, the scripts that run every experiment with our outputs, and download scripts for public datasets.

C COMPILING A POC INTO A TEMPLATE

C.1 Constructing the API-to-Operation Map

Each operation record of Π specifies how the compiler lowers the call (an HTTP exchange, a file operation, a process creation or a socket operation), the resulting event class in Σ_σ, and the argument that contains its operand. For each API, we run a no-op script and a script that calls the API under the host collector, normalize both event streams and take their multiset difference. We repeat this for several argument values and program states and keep the operations present in every difference.

C.2 Template Extraction

The compiler enumerates the execution paths through a PoC and resolves the target endpoint of each. It lowers each path in both run modes and emits every mapped call in program order.

C.3 Process Scoping and Path Enumeration

Before extraction, a sanitizer normalizes the PoC. It rewrites path-library and HTTP-session idioms into plain calls, substitutes command-line defaults, collapses exception handlers and keeps only statements needed by mapped calls. In its *call-chain* mode, the sanitizer keeps calls to functions defined in the PoC when their call chain leads to an anchor, and the compiler inlines those functions at their call sites. In its *plain* mode, it keeps only the function bodies. The compiler splits the syntax tree at every branch into straight-line programs, up to 256 per PoC. The compiler assigns program-scope calls to the PoC's process. A process-creating call introduces a new subject for the later calls in the created process. The distinct templates of both sanitizer modes and both run modes form a family. The compiler merges textually identical rules and removes rules without an edge. If it cannot resolve a request URL built by the code, it takes a URL from the PoC's usage text whose host is a placeholder, a loopback or a private address.

The compiler selects at most two literal fragments per PoC for payload rules, from the body, form, JSON, query or non-standard header arguments of its HTTP calls. A fragment passes the *fragment test* if it is at least eight characters long and contains a path traversal or one of `{}()$;`. Markup brackets do not count toward the test. Format placeholders in a fragment become wildcards. The compiler adds one payload variant of each request rule per fragment.

C.4 Emission Order

The compiler emits edges in program order along each straight-line path, and that order is the template's $\prec$. When logs contain no timestamps, we impose neither the order nor the window condition.

C.5 Release, Freeze and Template Details

We will release Π, Σ_σ, T, L, R^* (§4.2), the label normalization and the collector profiles (Table 1) under a commit ID and the SHA-256 hash of a manifest. If two fields embed the same variable in different text, as `"/user/"+x` and `"id="+x` do, their values differ for every x, and the fields share no symbolic variable. Neither do fields whose selectors are encoded differently. We replace each such field with a fresh anonymous wildcard that matches any value. We represent each vertex-label component as a normalized concrete value, a symbolic token shared by linked fields or a fresh anonymous token. A call is hidden from the compiler if its target is not syntactically fixed or if its call site or operands depend on monkey patching, reflection, dynamic imports, run-time callbacks or custom raw-socket clients. The compiler accepts every parsable PoC. Across the compiled PoC corpus, templates contain a median of 6 edges and 4 concrete vertices, and 93% have more than one edge.

C.6 Observability

We call an operation or vertex *structurally* observable if a deployed collector logs it, and a label component *field* observable if the log format includes it. We match each rule as a whole, and only on targets whose collectors record every required role and event class.

D ALIGNING TEMPLATES TO LOGS

D.1 Seeding Candidates

A candidate is the h-hop neighborhood of a process vertex in G_{evt}. We try candidates in turn and raise the alert at the first candidate with a valid mapping of the template.

D.2 Learned Candidate Filter

On large provenance graphs, Encore prunes candidates with a learned filter before refinement. The filter discards a candidate when the order violation $\text{viol}(z_{tem}, z_{cand}) = \| \max(0, z_{tem} - z_{cand}) \|_2^2$ between the template's encoding z_{tem} and the candidate's z_{cand} exceeds a threshold s. The encoder embeds each vertex from its type and hashed label tokens, passes messages over three layers with non-negative weights per event class and direction, and appends hashed counts of the vertex labels and of the labelled paths of up to two hops from the root. We train it with the order-embedding loss [56, 62] on 40,000 template–candidate pairs from the earliest part of the E3 CADETS time span, choose s, h and k on validation pairs from the following part, and freeze them for every test set. On these benchmarks, the candidates are the h-hop neighborhoods of the target vertices labelled with the template's rarest label. With

the filter, Encore discards up to 91 % of candidates and finds every template's true match across the full E3 test graphs. We compare the filter's accuracy with ProvG and Poirot in §5.8.

D.3 Timing and Staging

The emission order $\prec$ (Appendix C.4) is a linear extension of the edges' dependency constraints. We timestamp a matched template edge with $t(\mu_E(e))$, the time of its last event. If the match passes through intermediate process creations, the last event is on the edge that ends at the target vertex. We allow a clock skew of up to ϵ between collectors on the same target. During refinement, we require $t(\mu_E(e_i)) \le t(\mu_E(e_{i+1})) + \epsilon$ for consecutive edges in $\prec$, and $\max t - \min t \le \Delta$ over matched edges. Every edge of P_e before $\mu_E(e)$ is a process creation by the subject or its descendants. On edges of some classes, the path contains no intermediate creation. For an edge *into* the subject, such as an inbound protocol record, $P_e = \mu_E(e)$. For a process-creation edge, the prefix is a chain of creations, so an ancestor may spawn the child through intermediate shells. The bound k lets a match pass through process creations recorded in host logs between a PoC's operations. Distinct template edges must end in distinct events. No compiled rule is all wildcards.

D.4 Refinement and Labels

The matcher extends an injective mapping outward from the seed process. The matcher accepts a candidate label if it matches a disjunct of the template label. A placeholder for a command-line argument or a payload matches any label. A disjunct with a wildcard compiles to a pattern, and before matching it we normalize path separators and, on Windows paths, case. Any other disjunct must equal the label under the same normalization.

E EVALUATION DETAILS

E.1 Per-CVE Results

Table 5 lists the per-CVE results. We exploit the 2 Windows CVEs in a Windows 10 virtual machine with Process Monitor and Sysmon, against CompleteFTP 8.3.3 and WinRAR 7.12. The CompleteFTP PoC reads the credentials logged during first-run setup and opens the manager as administrator. In the WinRAR run, we record the exploit archive from its creation to its extraction, which writes the payload into the Startup folder.

For PrintNightmare, the PoC calls `par.hRpcAsync AddPrinterDriver` with a driver container and sets its configuration file to `winhttp.dll`, then `kernelbase.dll`, then the PoC's DLL in each `old` slot of the spooler's x64 driver directory. MS-PAR (the Print System Asynchronous Remote Protocol) specifies that the server copies the files named in a container, so we wrote Π's entry for that call as a write of those paths. The host rule matches any process that writes one of those paths, and in the logs it matches `spoolsv.exe` writing `evil.dll` into those slots.

E.2 Coverage over the PoC Population

The KEV catalog lists 1,186 CVEs disclosed between 1 January 2019 and our 30 September 2025 snapshot. Of these, 942 have a public PoC and 771 have a Python PoC. Our compiler parses 729 of the Python PoCs, and Π maps every log-relevant call in 651 of them. Compiling the 45 PoCs again yields byte-identical templates.

Table 5: Per-CVE results. *Source*: ■ Splunk's attack data, □ our run. *Found by*: P payload, else H host, else R request rule. *Encore-CodeQL*: ● finds it. *Splunk rule*: ● fires, ○ silent. *Text*: share of rule operands in the NVD description. Blank: none.

	2019-16116	2020-6287	2021-1675	2021-3129	2021-22205	2021-41773	2022-1388	2022-22947	2022-22963	2022-22965	2022-40684	2022-42889	2022-46169	2023-3519	2023-4966
Source	□	■	■	□	□	□	■	□	□	■	■	■	□	■	■
Found by	H	R	H	P	R	P	R	P	P	R	R	R	P	R	R
Encore-CodeQL				●	●		●	●	●	●	●	●	●		●
Splunk rule			●				●		●	●	●	●		●	●
Text	0/2		0/5	0/3	0/3	0/2	0/3	1/4	0/1	0/3	0/5	0/2	1/1	0/2	0/3

	2023-20198	2023-22515	2023-22527	2023-23397	2023-24489	2023-29298	2023-29357	2023-35078	2023-35082	2023-36844	2023-36845	2023-36846	2023-36847	2023-38646	2023-40044
Source	■	■	■	■	■	■	■	■	■	■	■	■	■	□	■
Found by	R	R	R		R	R	R	R	R	R	R	R	R	P	R
Encore-CodeQL	●	●	●		●		●	●	●	●	●	●	●	●	
Splunk rule	●	●	●	●	●	●	●	●	●	●	●	●	●		●
Text	0/3	0/5	1/3		0/8	0/3	0/4	0/4	0/2	0/1	0/1	0/1	0/1	0/2	0/3

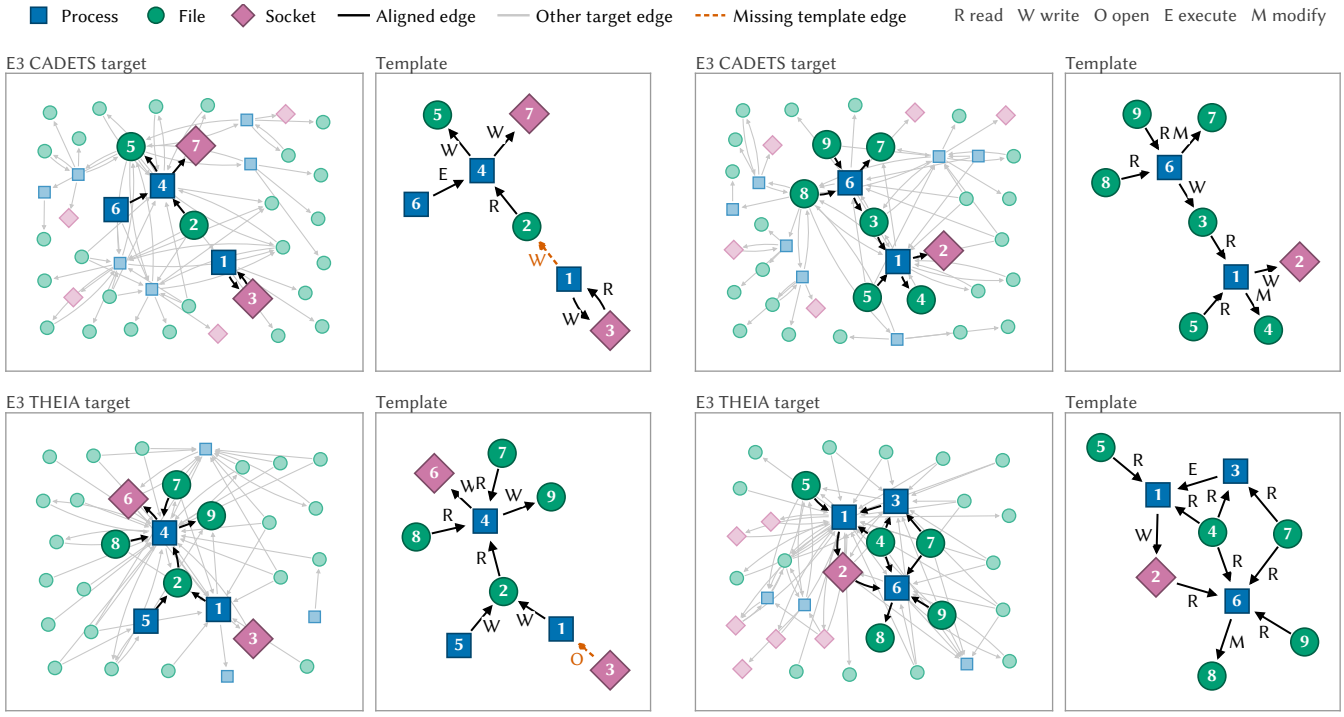
	2023-42793	2024-1708	2024-1709	2024-4040	2024-5806	2024-7593	2024-21683	2024-21887	2024-23897	2024-25600	2024-36401	2025-5777	2025-8088	2025-24813	2025-31161
Source	□	■	■	■	■	■	■	■	■	■	□	■	□	□	■
Found by	R	R	R	H		R	R	R	R	P	P	R	H	R	R
Encore-CodeQL	●	●	●	●		●	●		●	●	●	●		●	●
Splunk rule	○	●	●	●	●	●			●	●		●		●	●
Text	0/5	0/11	0/2	0/8	0/1	0/4	0/6		1/1	0/3	1/2	0/1		0/2	0/2

E.3 Setup Details

For each CVE, two of us confirmed that different people wrote the recorded and the compiled PoC, that their repositories share no fork or upstream, and that the recorded PoC copies no payload from the disclosure's write-up. We spent 5 person-hours per CVE on average on deployment, verification and recording. ESCU rules outside the comparison in §5.2 cover the remaining 3 instrumented web CVEs, CVE-2024-21887 and CVE-2020-6287. Splunk files the logs of the last two under CVE-2023-35081 and CVE-2025-31324, and the rule for CVE-2025-31324 fires on the logs of CVE-2020-6287. The rules of 23 families apply to the ATLASv2 host events, and those of 43 families apply to the NASA web log. The instrumented runs contain 147 requests and a median of 3,127 host events per CVE. The logs of each run show the exploit's effect. ATLAS [1] was evaluated on ten reconstructed APT attacks that exploit six distinct CVEs [43]. LLMCloudHunter [45], a Sigma rule generator, was evaluated on 20 annotated cloud threat reports.

E.4 Alert Localization and Host-Only Rules

Without using the matcher, we joined the proxy's side of each connection to the matching accept call on the server. Up to 12 workers shared a listening socket. In 4 runs, the exploit executes a command. All 42 processes started by the exploit descend from the serving process and began while that process handled the exploit request. For CVE-2022-22947, the rule compiled from the PoC contains a write and a read of the same route under `*/actuator/gateway/routes/*`. The logs show the two operations as two requests, a `POST` that creates the route and a `GET` that reads it back. Both requests are attached to the serving process, which spawns the attacker's



(a) Negative pairs

(b) Positive pairs

Figure 9: Sample (a) negative and (b) positive template–target pairs from ProvG's benchmark. ProvG classifies both negative pairs as matches, and Encore classifies all four pairs correctly. ProvG decides from learned embeddings, while Encore checks every template edge against the target and rejects each pair in (a), where the dashed template edge is missing.

command between them, and the alignment matches both there. The host-only rules match the driver DLL that the print spooler writes into System32 under PrintNightmare, the process chain of the CompleteFTP local privilege escalation, and the alternate-data-stream write and `rar.exe` process of WinRAR's CVE-2025-8088, whose exploit is an archive with a traversal path in an alternate data stream. For each host-rule alert, we record the machine that ran its operations. In Table 5, both a host rule and a request rule find CrushFTP's CVE-2024-4040.

E.5 SIEM Field Predicates

Defenders can run each rule as a SIEM correlation search with one field predicate per edge. We compared this form with the alignment on 40 CVEs. The field predicates return every logged event that satisfies any one of them, a median of 10 events per CVE and up to 600. For 15 CVEs, these events come from several processes. The alignment returns one connected match at the serving process, with a median of 2 events. We also drew queries from 7 datasets of DARPA Engagements 3 and 5 [15, 16] and from the endpoint detection and response (EDR) records of ATLASv2, in which every process instance is one vertex. Each query is a small subgraph of one process's events, and we paired it with a candidate that lacks the query. Of these 4,096 candidates, 45 % contain every operand of their query, and 219 contain all of its operations somewhere. Among 2,048 candidates with queries from other processes, 130 have all query operations. The predicates alert on all of these when run independently and reject them when joined on shared vertices.

E.6 Rule Evasions

AMIDES collects hand-crafted evasions of SIEM rules from enterprise networks, and SPECTRA generates families of transformations against rule syntax. The 11 transformations change header case, add junk parameters, reorder parameters, percent-encode parameter values and path segments, insert whitespace and comments, write a command as an absolute path, insert `/./` segments, append a path parameter and rename an XML namespace. We give Poirot a query graph of the same shape, labelled with the concrete values recorded in the unmodified run. Because Poirot finds no candidate alignment for its missed variants, it computes no score for them, and its counts are the same for every score threshold C_{thr} from 2 to 12. ProvG's results are the same whether we anchor its query at the highest-degree process or at the serving process.

E.7 Language-Model Prompt

We described the target's logs as web-server access logs, proxy logs with request bodies, and host process and file logs. We asked for rules for a remote run, a local run or both, excluding the attacker's machine. The worked example is a rule for a CVE outside the corpus. As with Encore, the models write rules before any exploit log exists. The full prompt for each CVE and the models' outputs will be in our artifact (Appendix B).