أكاديمية كاوست
KAUST ACADEMY

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

APHE Massar: An Attack-Path Hypothesis Engine for Exposure-Based Vulnerability Prioritization

Naif Alanazi, Abdallah Abonadi, Nawaf Alsahabi,
Osama Aldhayan, Azzam Balabaid, Prof. Mudassar Aslam

naif.alanazi@kaust.edu.sa, abdullah.abonadi@kaust.edu.sa, nawaf.alsahabi@kaust.edu.sa,
osama.aldhayan@kaust.edu.sa, azam.balabaid@kaust.edu.sa, maslam@brookes.ac.uk

Abstract

Vulnerability scanners routinely report hundreds of findings per host, yet a defender can only remediate a few at a time; the operative question is which weaknesses combine into a realistic attack, and which to fix first. We present **APHE Massar**, an Attack-Path Hypothesis Engine that ingests exposed assets and known vulnerabilities — from the National Vulnerability Database (NVD) and CISA's Known-Exploited-Vulnerabilities (KEV) catalogue — builds a typed attack graph, ranks the resulting attacker paths by an explainable likelihood-times-impact score, groups them into a defender-sized worklist, and attaches remediation quoted verbatim from the feeds. A central design decision separates concerns: deterministic code builds the graph and computes the ranking; the language model only judges which chains are plausible and writes their rationale — and we prove by ablation that the ranked order and scores are identical whether the model is absent, agreeable, or adversarial. Its one further job is drafting *detection* guidance, which a guard admits only when every concrete indicator it names traces back to the scan's own evidence; it never writes remediation. It runs offline on committed fixtures and end-to-end against three live hosts on two operating-system families — Metasploitable 2 (23 open ports, 431 findings, ranked in 0.7 seconds from the recorded scan) and both Metasploitable 3 guests, with no code change between them; it can scan an unseen host in 51 seconds. That run also tests our central claim honestly: where a host's exposed CVEs are uniformly high-severity and known-exploited the ranking coincides with a KEV-first sort, and our differentiation appears only through the privilege-and-chaining dimension — 2.3% of ranked pairs, decisive in the root-versus-foothold case a severity sort cannot separate. We therefore claim prioritisation *where the data supports chaining*, together with coverage honesty, explanation and actionable remediation, rather than an unconditional improvement over severity ranking. On the committed fixture it still produces the two-CVE chain — an internet-facing Log4Shell foothold that *enables* a Dirty Pipe local-root escalation, ranked (65.3) above the foothold alone (61.4). 900 automated tests pass, and we include the one label-free metric on which the engine does *not* beat a severity baseline.

Date: 9 August 2026
Code: Private repository (access on request)

Contents

1 Introduction 2

2 Related Work 3

3 Methodology 4

3.1 System overview . . . 4

3.2 From assets to a typed attack graph . . . 4

3.3 Scoring . . . 6

3.4 What the language model is allowed to write . . . 6

3.5 Evaluation procedure . . . 7

4 Results 8

4.1 The engine, end to end . . . 8

4.2 The interface a defender actually reads . . . 10

4.3 Is the ranking any good? An honest evaluation . . . 13

4.4 Verification and component status . . . 22

5 Discussion 23

6 Limitations and Future Work 24

7 Conclusion 27

References 27

1 Introduction

Automated vulnerability scanners and asset-discovery tools have made it easy to enumerate weaknesses, but hard to act on them. A single host can yield tens or hundreds of Common Vulnerabilities and Exposures (CVEs), presented as a flat list ordered — if at all — by an isolated severity score. Defenders, who have finite time, must nonetheless decide *which few of those weaknesses matter*. What a severity list omits is precisely what an attacker relies on: the way individual weaknesses *chain*. A medium-severity local privilege-escalation is unremarkable in isolation, but decisive once a separate flaw has already granted a foothold on the same host.

This project addresses that gap. Given a description of exposed assets and the vulnerabilities affecting them, APHE — an Attack-Path Hypothesis Engine — *hypothesises* and *ranks* the attacker paths through those assets, so that a defender can see which chain is most dangerous and in what order to remediate. The engine is named APHE throughout this report; the dashboard a user actually opens carries the product name *Massar*, which is what appears in every screenshot in Section 4.2. They are the same system, and we use the shorter *APHE* from here on. The contribution is not detection — any scanner lists vulnerabilities — but the surrounding judgement a flat list withholds: deciding which weaknesses combine into a path, ranking those paths *where the chaining structure gives severity something to differentiate*, collapsing the result into a defender-sized worklist, and attaching remediation. We are deliberate about the qualifier. Running the engine against a real host (Section 4) shows that when every exposed CVE is uniformly high-severity and known-exploited, the ranking coincides with a KEV-first sort; the engine earns its place through the privilege-and-chaining dimension, which acts only when chains reach different privilege levels. We report that boundary rather than claim an unconditional improvement over severity ranking.

Two assumptions bound the work. First, APHE hypothesises and ranks attacker paths; it does *not* execute exploits — validating a hypothesised path (proof-of-concept or red-team) is a separate, manual activity outside this engine. Second, all development and testing run in an isolated laboratory against assets we own and intentionally-vulnerable targets, so the exercise is authorised by construction; a scope allowlist enforced in code keeps the tooling on that range. The engine reasons about multiple hosts, but models each as reached directly from the internet: host-to-host reachability is represented and provenance-tagged, yet never *measured*, so APHE does not claim a pivot from one machine to another (Section 6).

APHE is structured as a seven-stage pipeline — gather, enrich, vulnerability lookup, graph construction, hypothesis, scoring, and dashboard — in which each stage consumes and produces typed records (Section 3, Figure 1). Its guiding *design* principle is a strict separation between computation and judgement: deterministic Python builds the attack graph and computes the ranking, so the output is reproducible and explainable, while the fuzzy sub-task of judging which hypothesised chains are plausible is delegated to a language model whose every proposed path is validated against the real graph. The same principle decides what else the model is allowed to write. It answers *how would I see this?* — detection and observability guidance — because no authoritative source supplies that text and a deterministic guard can check the result against the scan's own evidence. It does not answer *how do I fix this?*, because CISA and NVD already answer it and paraphrasing an authoritative remediation directive can only introduce a new way to be wrong (Section 4.2).

That design is now realised end-to-end. The deterministic half — graph construction, the privilege-labelling that drives chaining, scoring, worklist grouping, and remediation extraction — runs and is tested. The language-model half is integrated: on a run with a provider configured, a single structured call judges each enumerated candidate and writes its rationale, and the mode of

every run is recorded in its output so a reader always knows whether the model was consulted. Crucially, the model changes explanation, never rank: we prove by ablation (Section 4) that the ranked order and scores are identical whether the model is absent, calls every path plausible, or calls every path implausible. The `gather` stage now ingests real `nmap` output and has been run against three live hosts on two operating-system families, while `enrich` resolves CPEs from the scanner's own identifiers, then a small curated table, with the NVD CPE dictionary correcting the result where a resolver is configured. The stages that remain partial are stated where they arise and gathered in Section 6.

2 Related Work

Attack graphs from vulnerability data. Attack graphs — models of how an attacker can chain weaknesses toward a goal — have a long lineage, generated early on by model checking [1] and by logical inference over network configuration [2], and surveyed comprehensively by Kaynar [3]. A strand directly relevant to APHE builds such graphs automatically from public vulnerability data: Aksu et al. derive attack graphs from the National Vulnerability Database, mapping CVSS fields to attacker pre- and post-conditions [4]. On the input side, internet-wide scanning and search engines such as Censys [5] and attack-surface metrics [6] have made exposed-asset data widely available. APHE follows this NVD-grounded, exposure-driven tradition, but targets the *prioritisation* of the resulting paths rather than their exhaustive enumeration.

Representing how CVEs chain. A recurring difficulty is that a raw CVSS score describes a vulnerability's severity but not how it *chains*: it does not state which privilege an attacker must already hold, nor which they gain. Poisson et al. address this with CAPG, a representation whose nodes are (machine, user) attack positions and whose edges are CVE exploitations that move the attacker between positions [7]. APHE adopts a deliberately lighter form of this idea (Section 3): it labels each CVE with a privilege precondition and postcondition derived from its CVSS vector and CPE part, and admits a chaining edge only when one CVE's postcondition satisfies another's precondition.

Large language models for offensive security. Recent work applies LLMs to penetration testing. PentestGPT reports that the dominant failure mode is the loss of long-horizon context and mitigates it with an explicit task-tree state [8]. Incalmo raises the level of abstraction the model is asked to work at — planning in high-level declarative tasks that domain-specific agents then execute, rather than emitting low-level actions directly — and reports acquiring critical assets in 37 of 40 emulated multi-host networks, against 3 of 40 for existing LLM-assisted offensive systems [9]. Others let a model generate attack graphs directly, optionally with retrieval augmentation [10, 11]. The consistent lesson is that an LLM should reason *over* a grounded, structured representation rather than invent structure from prose — the principle APHE encodes by confining the model to judging plausibility over a deterministically-built graph.

Scoring and prioritisation. Attack-graph research has long combined per-step likelihoods into path-level risk [12, 13], and the Exploit Prediction Scoring System (EPSS) supplies data-driven per-CVE exploitation probabilities [14]. APHE uses a deliberately simple, explainable likelihood-times-impact score (CVSS severity, a known-exploited multiplier, and a per-hop penalty); EPSS is noted as a future signal but not yet incorporated.

Choosing which fixes to apply. Ranking paths and choosing remediations are different problems, and the second has its own literature. Noel et al. formulate minimum-cost network hardening over exploit dependency graphs, reducing the choice of which initial conditions to disable to a set-covering problem and solving it approximately [15]; Poolsappasit et al. later cast remediation selection as optimisation under resource constraints on a Bayesian attack graph [13]. APHE's

"highest-coverage fix" (Section 3) is a deliberately plain instance of this idea: a single-pass greedy selection of the action that removes the most enumerated chains. **We claim no algorithmic novelty for it.** It is standard greedy set-cover applied to a graph our own pipeline produced, and it is not even the adaptive variant, which would score better on coverage metrics but is not the code a defender runs. Its contribution here is integration rather than technique: it is computed over live KEV- and NVD-derived exposure for a real host, and its effect is reported as a directly measured count of chains removed rather than as a proxy such as node centrality.

The gap. Prior work supplies the pieces — NVD-derived attack graphs [4], an attack-position chaining model [7], evidence that LLMs must be grounded in structure [8, 9], and probabilistic path scoring [12] — but does not combine them into an exposure-driven *prioritisation* tool in which deterministic code owns the graph and scoring while a language model is restricted to a grounded plausibility judgement. That combination is the gap APHE targets.

3 Methodology

3.1 System overview

APHE is organised as a seven-stage pipeline (Figure 1). The *gather* stage collects exposed-asset data; *enrich* normalises each service into a Common Platform Enumeration (CPE) identifier; *vulnerability lookup* matches CPEs to CVEs and flags known-exploited ones; *graph construction* assembles assets and vulnerabilities into a typed attack graph; the *hypothesis* stage enumerates candidate attacker paths; *scoring* ranks them; and the *dashboard* presents the graph and the ranked paths. Every stage consumes and produces `pydantic`-typed records, so a mismatch at a stage boundary fails loudly rather than propagating silently.

The design is governed by one principle — a separation of *computation* from *judgement*. Deterministic Python performs all work with a defensible correct answer: CPE/CVE matching, graph construction, and scoring. The single fuzzy sub-task — deciding which hypothesised chains are *plausible* — is the role of a language model, constrained to judge and explain candidate paths that already exist in the graph. Candidate-path enumeration proceeds deterministically from the graph and scoring ranks the enumerated paths; the language model, when a provider is configured, judges each candidate's plausibility and writes its rationale, *layered on top of* the enumeration rather than replacing it, and — by construction and by an ablation test — without altering the rank. The description below separates the mechanism (what the graph and score compute) from the parts that remain partial (Section 6).

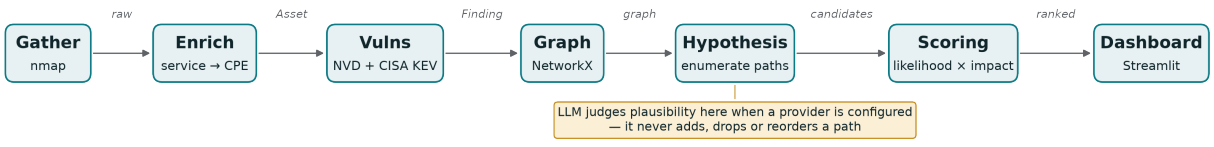


Figure 1 The APHE pipeline. Exposed assets are enriched to CPEs, matched to CVEs (with CISA KEV flags), assembled into a typed attack graph, hypothesised into candidate paths, and ranked. Deterministic stages own the graph and the score; the hypothesis stage is where a language model judges plausibility (when a provider is configured), without changing the deterministic rank.

3.2 From assets to a typed attack graph

Enrichment and vulnerability lookup.. Each service is mapped to a CPE 2.3 identifier, e.g. `cpe:2.3:a:apache:solr:8.11.0:*:*:*:*:*:*:*`, whose fields record the part (a application,

`o` operating system, `h` hardware), vendor, product, and version. Vulnerability lookup then matches a CPE to CVEs by *version range* rather than by exact string: a CVE that lists an affected range $[8.6.0, 8.11.1)$ matches version `8.11.0`, which an exact-string table would miss. Each match is annotated with its CVSS score and flagged if it appears in the CISA Known-Exploited-Vulnerabilities (KEV) catalogue. Following Aksu et al. [4], all inputs are drawn from public data (NVD, KEV); no exploitation is performed.

Nodes, edges, and privilege labels.. The graph has four node types — an internet entry point, hosts, services, and CVEs — with CVE nodes *host-qualified* so that the same CVE on two hosts remains two distinct positions. A service-borne CVE attaches to its service by a `has_vuln` edge (reachable over the network); a host-scoped CVE, such as a kernel flaw obtained from OS fingerprinting, attaches to its host by a `has_local_vuln` edge (reachable only once local access is held). This distinction is the crux of realistic chaining. Inspired by the attack-position model of Poisson et al. [7] but using only public data, each CVE is labelled with a privilege *precondition* and *postcondition* derived deterministically from its CVSS vector (Privileges-Required, Integrity) and CPE part; the labels take values on an ordered ladder (NONE < APP-USER < APP-ADMIN < OS-USER < OS-ADMIN).

The `enables` edge and entry-admissibility.. A directed `enables` edge is added from CVE A to CVE B on the same host exactly when A's postcondition satisfies B's precondition and the privilege strictly increases; the strict increase guarantees the chaining subgraph is acyclic. A separate *entry-admissibility* rule marks a CVE as a valid starting point only if it needs no prior privilege and is network-reachable on an exposed host. Path enumeration keeps only chains that begin at an entry-admissible CVE, which correctly discards the nonsensical path of reaching a purely local privilege-escalation directly from the internet. This enumeration is deterministic. When a provider is configured, a language model then judges each candidate's plausibility and writes its rationale on top of this enumeration — without adding, removing, reordering, or re-ranking any path, a non-influence on rank we verify by ablation (Section 4).

Two mechanisms that keep the output honest.. Separating computation from judgement is only useful if a reader can *see* which is which, so two properties are carried in the data rather than left to prose. First, **every edge records its own provenance**, drawn from a fixed vocabulary: `observed` (asserted by the scan input), `measured` (actively enumerated in the laboratory), `assumed` (a heuristic, not a measurement — the escalation rung above is the main case), and `derived` (computed from CVSS privilege labels, with no exploit run). The graph builder stamps it; the dashboard draws each class in its own colour and line style, names the class when an edge is selected, and prints the run's whole census in a strip beneath every page, so a guessed edge cannot be mistaken for a measured one. On the real host this matters quantitatively: 5,289 of 5,745 edges are `derived`, which is a fact a reader is entitled to know before trusting the picture. The obligation is enforced rather than intended: a test asserts that no synthesised edge may be drawn as `observed`, which was added after two edges — the internet-to-host exposure heuristic and the synthesised privilege-grant edge — were found being drawn in the observed style on every host, under an inspector that described them as facts rather than inferences.

Second, **model-written text is marked as such**. Each ranked path carries the source of its rationale — computed template or language model — alongside the model's plausibility verdict, as structured fields rather than as a prefix inside the prose. An earlier version signalled lower confidence by prepending a phrase to the rationale string, which put the one honesty marker on the card *inside* text the model itself wrote, where any rewording would silently erase it. A reader therefore never has to guess whether a sentence came from the scoring arithmetic or from

a model.

3.3 Scoring

Each candidate path p receives a risk score combining impact and likelihood. Impact is the maximum CVSS base score $s_{\max}(p)$ along the path, raised by a multiplier m when the path's terminal CVE grants operating-system administrator privilege (so a chain reaching root outranks a foothold of equal base severity). Likelihood starts at 1 and is multiplied by an *exploitability* factor $e(p)$, by an exposure factor w_{net} if the entry host is internet-facing, and by a per-hop penalty over the h edges of the path. Exploitability treats evidence of real-world usability as a spectrum rather than as membership of the KEV catalogue alone: a CVE listed by CISA as known-exploited contributes w_{kev}, and a trivially-reachable critical remote-code execution — entry-admissible and scoring at least θ — contributes w_{rce}. These are two readings of one underlying signal, so they are combined by *maximum* rather than by product, which prevents a flaw that satisfies both from being counted twice:

$$e(p) \;=\; \max\!\left(w_{\text{kev}}^{[\exists c \in p\,:\,c \in \text{KEV}]},\;\; w_{\text{rce}}^{[\exists c \in p\,:\,\text{adm}(c)\,\wedge\, s(c) \geq \theta]}\right), \tag{1}$$

where $[\cdot]$ evaluates to 1 when the bracketed condition holds and 0 otherwise, so an unsatisfied condition contributes a neutral factor of 1, and $\text{adm}(c)$ marks a CVE admissible as a chain entry point. The path score is then

$$\text{score}(p) \;=\; 100 \cdot \min\!\left(\frac{s_{\max}(p)\, m \;\cdot\; e(p)\, w_{\text{net}}\, (1-\rho)^{h}}{R},\; 1\right), \tag{2}$$

normalised against a privilege-neutral constant $R = 10\, w_{\text{kev}}\, w_{\text{net}}$ so that a single-CVE foothold keeps a fixed reference value. Because $w_{\text{rce}} < w_{\text{kev}}$, a KEV-bearing chain and the normalisation baseline are both numerically unchanged by Equation 1; the term lifts only critical remote-code execution that CISA has not listed. With the committed configuration ($w_{\text{kev}} = 2.0$, $w_{\text{rce}} = 1.6$, $\theta = 9.0$, $w_{\text{net}} = 1.5$, $\rho = 0.15$, $m = 1.25$, $R = 30$), the vsftpd 2.3.4 backdoor — CVSS 9.8, not in KEV, reaching root over three edges — scores $100 \cdot (9.8 \cdot 1.25 \cdot 1.6 \cdot 1.5 \cdot 0.85^3)/30 = 60.2$, matching the ranked output in Section 4. All weights ($w_{\text{kev}}, w_{\text{rce}}, \theta, w_{\text{net}}, \rho, m$) are read from a configuration file, not hard-coded, so the ranking policy is tunable and auditable. The design follows the path-probability tradition [12] while remaining deliberately simple and explainable; EPSS [14] is left as a future likelihood signal.

3.4 What the language model is allowed to write

The separation of computation from judgement decides not only whether the model may rank, but what text it may author at all. Two questions sit beside every finding. *How do I fix this?* already has an authoritative answer — CISA's required action for a known-exploited CVE, NVD's fixed-version bound, and a containment step computed from the attack graph — so APHE copies that answer verbatim and attributes it, and the model is excluded from the fix block entirely. *How would I see this?* has no authoritative source, and before this layer it was answered for none of our findings, so it is the job we give the model.

The layer has two rungs, and the first never depends on the second. The deterministic rung is a curated table of class-level guidance keyed by CWE: 44 hand-written entries, each naming where to look, how reliable the signal is, and what the weakness class looks like in telemetry. It needs no provider, no key and no network, and it resolves for 46.4% of the findings on the evaluation host. Only the residue reaches the model, which is asked for a per-CVE draft: either a detection rule, or an explicit statement that it could not draft one from this evidence.

What makes such a draft admissible is a deterministic *guard* that runs before anything reaches the screen. The guard first assembles the evidence the scan itself produced — service, port, versions, the CWE list, the attack vector, and any ports or file paths cited in NVD's own description — and then applies seven checks. It refuses a network-surface rule for a locally-exploitable finding; requires the output to be structurally valid with a resolvable condition; requires its log sources and fields to come from a committed vocabulary; requires a stated reason when the answer is that no rule was possible; and — the check the layer exists for — requires that *every concrete indicator named inside the rule trace back to that evidence*. A rule naming a port, path or string the scan never saw is rejected outright, not shown with a caveat. A seventh check requires the rule to name at least one concrete, non-wildcard value; it was added after a rule whose every field was `"*"` passed the other six — nothing in it was unsupported because nothing in it was asserted.

The worked example is the evaluation host's top-ranked finding. NVD's own description of CVE-2011-2523 states that the backdoored vsftpd 2.3.4 archive "opens a shell on port 6200/tcp"; the evidence extractor records `6200` as a cited port; the model's draft selects on destination port 6200; and the guard accepts it because that single indicator traces to the feed's sentence rather than to the model. Had the draft named port 4444 instead, the same check would have rejected it.

Two limits are deliberate, and are stated on screen as well as here. The guard validates indicators *inside the rule*; the free-text watch items presented alongside it are not evidence-checked, and are labelled as unchecked suggestions. And a guard verdict is a statement about grounding, not about correctness: it does not establish that a rule fires on real traffic, that it is specific enough to be quiet, or that the necessary telemetry is collected at all. The strongest claim this layer supports is therefore an evidence-grounded detection *hypothesis*, which is why the interface never calls it a validated detection. The rule format is a variable rather than a commitment: the emitted record names its own format, so adding a second is a new format plus a new validator — and a format the guard cannot check is not admitted, because an unvalidated rule shown beside a validated one under the same footer is precisely the failure this design exists to prevent.

3.5 Evaluation procedure

Tools.. The system is implemented in Python 3.11+ (developed and tested on Python 3.12) using `networkx` for the graph, `pydantic` for the typed contracts, `requests` with `requests-cache` and `tenacity` for the NVD and KEV clients, and `streamlit` for the dashboard, whose graph views are drawn directly in the browser from the engine's own node and edge records rather than by a general-purpose graph library. A provider-agnostic LangChain factory — validated with Google Gemini, with a local Ollama backend supported as an option — drives the integrated language-model stage, selected at run time by `APHE_LLM=auto|on|off` and falling back to deterministic behaviour when no provider is configured. Tests use `pytest` and linting uses `ruff`; all dependency versions are pinned in a committed lock file.

Test inputs.. To keep the pipeline and its tests fully offline and deterministic, the demand for live scanning and live API calls is replaced by committed fixtures: an NVD-shaped CVE file, a KEV-shaped catalogue file, and a single-host reconnaissance record describing a web server (Apache Solr on a vulnerable Linux kernel). This fixture encodes a two-CVE scenario — an internet-facing Log4Shell (CVE-2021-44228) foothold and a Dirty Pipe (CVE-2022-0847) local privilege-escalation — exercising the full matching, chaining, and scoring path without touching a network.

Metrics.. Progress is measured in three ways. First, an automated test suite checks the risky logic (version-range matching, privilege labelling, graph construction, and the scoring arithmetic),

with the demonstration scores pinned as regression locks. Second, the end-to-end run is judged by the correctness of its output: the two expected findings are produced, the `enables` chain is formed, the bogus direct-to-kernel path is filtered, and the paths are ranked as expected. Third, because a prioritisation tool has no single accuracy number, results are read against two baselines: a conventional scanner, which emits an unranked flat list with no notion that the two CVEs chain, and exact-string CPE matching, which misses in-range versions such as `8.11.0`.

4 Results

4.1 The engine, end to end

End-to-end run.. Running the pipeline on the committed single-host fixture, entirely offline, produces the expected two-CVE attack chain. Enrichment yields two assets (the Apache Solr service and the host's Linux kernel), vulnerability lookup produces two KEV-flagged findings, and graph construction assembles a five-node, five-edge graph. Path enumeration and scoring return the two ranked paths of Table 1: the privilege-escalation chain that reaches root ranks first, above the foothold alone.

Table 1 Ranked attacker paths for the demonstration host. A conventional scanner would list both CVEs unranked, with no indication that they chain; APHE ranks the root chain above the foothold and attaches the enabling edge.

Rank	Path	Terminal privilege	Score
1	internet → host → Solr:8080 → Log4Shell → Dirty Pipe	OS-admin (root)	**65.3**
2	internet → host → Solr:8080 → Log4Shell	OS-user (foothold)	61.4

Correctness of the core mechanic.. Three properties are verified against the run. Version-range matching finds CVEs whose affected range contains the asset's version — Solr `8.11.0` matches Log4Shell's $[8.6.0, 8.11.1)$ — where an exact-string table listing a single version would miss it (the matching baseline). The `enables` edge forms exactly when the privilege conditions hold, producing the Log4Shell → Dirty Pipe link. The entry-admissibility filter discards the direct internet-to-kernel path, so no ranked path reaches the local privilege-escalation without first traversing the foothold. The two demonstration scores (65.3 and 61.4) are pinned as regression tests so they cannot drift silently.

Running against a real host.. Beyond the fixture, the engine has been run end-to-end against a live Metasploitable 2 target (172.20.10.3, `nmap` 7.99). The scan reports 23 open ports, which `enrich` resolves into 24 assets: 7 assessable (a CPE with a concrete version), 6 identified but version-undetermined (the scanner reports, for example, Samba as "3.X – 4.X"), and 11 unidentified. This three-way split is deliberate — "we found nothing" has three distinct causes a defender must be able to tell apart, and conflating them would let "no attack path" read as "secure". The interface splits the assessable bucket once more, into assets that produced CVEs and assets that produced none (§4.2), for a reason the enrichment stage cannot see. Vulnerability lookup returns 431 findings; notably, 191 of them are labelled for chaining only through a CVSS v2 fallback, because NVD never back-filled CVSS v3 onto most of this host's 2004–2012 CVEs, so without that fallback they would be silently unchainable. Graph construction and ranking complete in 0.7 seconds end-to-end, and the 89 enumerated chains collapse to a worklist of 12 prioritised outcomes, which the dashboard presents under the 6 entry points they share (§4.2). The full run originally required

a live NVD key, but the exact NVD and KEV responses for the host's assessed CPEs are now committed as a keyed sidecar, so the real-host result — all 431 findings and the ranked worklist — reproduces offline with no API key on a fresh checkout.

Three hosts, two operating-system families.. That target is the evaluation host, but it is not the only one. The engine has since been run against both guests of Metasploitable 3 — an Ubuntu 14.04 machine and a Windows Server 2008 R2 machine — *with no code change*, and both are committed with their own recorded feed responses so they replay offline (Table 2). Services resolve and are assessed on every platform tested, Microsoft IIS and PHP included, and on all three hosts the top-ranked chain reaches root.

Table 2 The three real hosts, all replaying offline with no API key. The engine is unchanged between them; only the recorded scan differs.

	Metasploitable 2	MS3 Ubuntu	MS3 Windows
Assets (assessed)	24 (7)	8 (5)	19 (6)
Findings	431	130	295
Graph nodes / edges	457 / 5,745	140 / 242	316 / 491
Chains → worklist	89 → 12	70 → 6	178 → 6
Top-ranked finding	vsftpd 60.2	Apache 59.4	PHP 63.9

Two practical observations came out of adding those hosts. First, *scan depth dominated everything in the engine*: `nmap`'s default top-1000 port list reaches only 11 of the 17 open ports on the Windows guest, and the six it misses include `8484` (Jenkins) and `8585` — the single port carrying 289 of that host's 295 findings. A default scan of that guest therefore yields 6 findings and 4 candidate chains, against 295 and 178 for a full-port scan. Second, the pipeline no longer depends on a prior recording pass. Given a target instead of a fixture it performs the scan itself — through the same scope gate — and queries the live NVD and KEV feeds, completing in 51 seconds on a host it had never seen and returning a result identical to the recorded replay of that host. Live runtime is dominated by the scan profile rather than by the engine: the same live path on the Windows guest, scanned full-port, takes 278 seconds and reproduces that host's committed fixture exactly.

The confirmed instant-root backdoor ranks first.. The real host's #1 is the vsftpd 2.3.4 backdoor (CVE-2011-2523) at score 60.2 — a trivially-exploitable RCE that opens a root shell, and the most-exploited service on the box. Two deterministic refinements put it there. First, an *exploitability* term: a non-KEV critical RCE is treated as a strong likelihood signal (a `critical_rce` multiplier, combined with KEV as a *maximum* so KEV chains and the demonstration numbers are unchanged to the decimal); the backdoor predates CISA's KEV list, so a binary KEV flag alone had buried it beneath KEV-listed mediums. Second, *direct-root* labelling: because vsftpd runs as root, its RCE *is* root, so the chain reaches root in one hop rather than through a hypothesised escalation. Which CVEs run as root is domain knowledge a CVSS vector cannot supply, so a committed file records them — a language model may *propose* additions offline, but a human commits them and the pipeline reads only the file, keeping the ranking deterministic and invariant to the model. Selecting the service on the dashboard's host map shows the CPE the engine actually resolved, which is how the vendor correction that recovers this CVE becomes checkable rather than asserted (Figure 3).

The same screen makes the central claim visible without a single number being explained. On this host the six entry points are ranked *vsftpd :21* at 60.2 with **3** CVEs, then Apache :80 (58.7, 86), ISC BIND :53 (52.2, 52), PostgreSQL :5432 (52.2, 59), OpenSSH :22 (51.2, 49) and *MySQL :3306 at*

*19.2 with **182***. MySQL carries 42% of everything found on the host and ranks last; vsftpd carries three findings and ranks first. Count is not risk, and the ordering says so on its face.

What a heterogeneous host exposed about identification.. The Windows guest was the first target whose software stack the engine had not already been tuned against, and it exposed three identification faults that a single-host evaluation could not have surfaced. All three share a shape: the tool named the wrong software.

First, `nmap` emits exactly one CPE per port, describing the process that is listening; anything running *behind* that process appears only as prose in the scanner's `extrainfo` field. Port 8585 is `Apache httpd 2.2.21` with the banner `(Win64) PHP/5.3.10 DAV/2` — and PHP 5.3.10 carries 236 CVEs that were entirely invisible, because the only identifier for that port belonged to the web server. Second, the scanner sometimes names the wrong product outright: it reports the Jenkins instance as `Jetty winstone-2.8`, a version string no release of Jetty ever carried, which matches zero CVEs.

The third was the most consequential, and it was a defect in our own graph construction rather than in the scanner. Findings were attached to service nodes by matching host and port and taking the first asset found, so once PHP was identified as a distinct asset on port 8585 the Apache node absorbed *all* 289 findings there, 225 of them PHP's. The top-ranked recommendation therefore told a defender to patch Apache in response to a KEV-listed *PHP* remote-code-execution flaw (CVE-2012-1823). Since every finding records the CPE it was produced from, that identifier now decides ownership. Correcting these took the host from 70 findings and 32 chains to 295 and 178, and moved the lead recommendation from "Apache `:8585`" to "PHP `:8585`, removing 146 of 178 chains".

This carries a caveat we state explicitly: *assessed* does not mean *correctly assessed*. Two of the six assessed CPEs on that host returned zero CVEs not because the software is sound but because the scanner's own identifier was wrong — `oracle:jsp:2.3` for a GlassFish server, `mortbay:jetty:winstone-2.8` for Jenkins. For a period our three-way coverage split could not distinguish those from a genuinely clean service, which is precisely the "no attack path reads as secure" failure the split exists to prevent, one level down. Coverage therefore reports a fourth bucket: assets that were assessed and returned nothing. It is measured on every host — two of six here, one of five on the Ubuntu guest, one of seven on Metasploitable 2, and zero on both synthetic fixtures — and it is drawn as a solid rather than a hatched segment, because the engine did look; what it cannot tell the reader is whether a zero means a clean service or a lookup under the wrong name, and the interface says so rather than implying the former.

4.2 The interface a defender actually reads

The pipeline's output is a ranked list, but a list is not yet a decision, and the interface is where the reduction either becomes legible or does not. The dashboard is therefore organised as six pages that answer one question each — *Home* (the front door, which deliberately states no figure and loads no run, Figure 2), *Host map* (what is exposed, Figure 3), *Attack map* (how it connects, Figure 4), *Results* (every way in, ranked, with both its remediation and its detection guidance, Figure 5), *Summary* (the whole host at once, Figure 10), and *About* (what the tool claims and refuses to claim). The properties below are reported because each was measured against the committed hosts rather than chosen on taste.

Grouping is a partition, not a filter.. The twelve prioritised outcomes on Metasploitable 2 are six services each reaching two privilege levels, so they are presented under the six *entry points* they share; every enumerated chain remains reachable from the page, and the lead entry states

its second outcome inline ("root 60.2, also reaches local user 22.7") rather than hiding it. Each entry in the ranked strip is labelled with its score *and* its finding count, which is what makes the ordering self-evidently not a severity sort: on this host the strip reads $60.2 \cdot 3$ CVEs at rank 1 and $19.2 \cdot 182$ CVEs at rank 6.

Two orderings that disagree, and the interface says so.. The single highest-coverage fix removes 30 of 89 chains — a 34% return that reads as poor — while a greedy residual plan shows that three changes clear 85% (30 → 54 → 76, then 84 → 87 → 89). Those two orderings genuinely disagree, and the page states the disagreement rather than letting a reader assume agreement: the vsftpd backdoor is first by risk and fifth by coverage, and Apache :80 leads on coverage. A selector lets a defender apply any subset of the plan and reports what survives, because greedy order is not interchangeable with an arbitrary one: the plan's first three changes remove 76 of 89 chains, while the other nineteen three-service combinations remove between 13 and 62. When a chosen subset leaves nothing standing the page reports that as a warning rather than as "secure", because the coverage bar still bounds what was examined at all.

The raw graph was retired on evidence.. An earlier version rendered the whole 457-node graph behind a toggle. Measured in a browser, that document is 2.3 MB and costs roughly eight seconds of synchronous main-thread work, and 5,289 of its 5,745 edges are the `derived` chaining edges — so the picture a reader spent eight seconds on was 92% inference. It is now replaced by two views that carry the same information at a density a reader can use: a per-service cluster whose CVE list opens in an inspector (so every CVE remains reachable), and plain sentences stating which service-to-service escalations *exist*, never how many edges back them (Figure 7).

Reporting those aggregates as *counts* was tried first and discarded, for a reason worth stating because it generalises. Aggregating Metasploitable 2's 5,289 `enables` edges by the service at each end yields 25 service-pair cells, and **23 of the 25 are complete bipartite** — every participating source CVE links to every participating target CVE — so a cell's value is the product of two set sizes rather than a count of independent facts. The largest reads 2,310, which is 165×14. Worse, shading such a cell by magnitude would assert an ordering the engine never computed and which contradicts the one it did: MySQL :3306 touches **4,321 of the 5,218** service-to-service edges (83%) and is the *lowest*-ranked entry point on the host. The rule we drew from it is general — before drawing a derived quantity, establish whether it is a fact or an artifact of set sizes.

Coverage, provenance and hypothesis framing are permanent furniture.. A strip beneath every page states the run's provenance census and its coverage split at once — on Metasploitable 2, *454 observed, 5,289 derived, 2 assumed*, and *7 of 24 checked, 1 found nothing, 17 never examined* — so a reader cannot reach a ranked list without meeting what was never examined. Unexamined assets are drawn hatched rather than coloured, because an unexamined asset is unmeasured, not clean, and the host map draws them as an explicit node rather than omitting them (Figure 3).

Model-written text is marked, and it is allowed to disagree.. Rationale prose carries a badge naming its source, and the badge also carries the model's verdict. This is not decorative: on a live scan of a containerised web service the model judged the top-ranked path *implausible* — "the local privilege escalation step assumes a Windows 11 OS that could not be determined and lacks a matching CVE" — and the engine ranked it first regardless (Figure 8). The disagreement is shown to the reader and changes no number: the ablation guarantee proved in §4.3, visible on screen rather than only in a test. A model that is permitted to disagree in public, and still cannot move a rank, is a stronger demonstration of the separation than one that always agrees. The remediation block beneath it states its own provenance in the opposite direction: *copied verbatim from CISA*

KEV and NVD, computed from the attack graph, or taken from a reviewed configuration file — no model wrote any of it.

The second question, and the only place the model writes for the reader.. Beside *how do I fix it?* the ranked card now answers *how would I see it?*, and the two sit side by side precisely because their provenance is opposite (Figure 5). The mechanism is described in Section 3.4; what it yields is this. The deterministic class-level rung resolves for **534 of the 856 findings (62.4%)** across the three hosts — 46.4% on Metasploitable 2, 77.7% and 79.0% on the two Metasploitable 3 guests — with no provider, no key and no network, so the free half of the feature survives a run with the model switched off entirely. That guidance carries its own strength on screen rather than in a footnote: of those 534, **234 are labelled weak** and 77 strong, because a class whose signature is not worth writing should say so instead of looking identical to one that is (Figure 9).

The model is scoped to the residue, and the measurement that justifies it also constrains it. Over a stratified sample of 90 answers drawn across the three hosts, 93.3% were structurally valid; of the concrete indicators examined inside proposed rules, seven were not present in the scan's evidence, and *the guard rejected every one of them*. Only a small minority of findings — on the order of three percent — yield an accepted rule at all. The common honest answer is telemetry guidance rather than a rule, and we report that instead of tuning for a higher rule count, because a high rule yield from thin evidence is a symptom rather than an achievement. On the same sample the model is also not merely paraphrasing the free table above it: the mean word overlap between an accepted answer and its corresponding class-level text is 7%, no answer exceeds 50%, and 56.5% name something specific to this host — a service, a port, a file — that a class-level entry could not.

We are explicit about what that does and does not license. This layer produces an *evidence-grounded detection hypothesis*. It is not a validated detection, nothing in this project reaches that bar, and the interface says so in the same box: the rule has not been tested against real telemetry or exploit traffic. The one measurement that would settle whether these hypotheses are *useful* is a human rating by a detection engineer, and it is deliberately not reported here because it has not been carried out (Section 6). Structural validity and practitioner usefulness are two numbers, and quoting the first as if it were the second is the specific overclaim this design is built to avoid.

Every CVE is reachable, and the page says what it is showing.. Selecting a service opens the complete list of its findings with a detail panel per CVE — 86 rows for Apache `:80` on the evaluation host, 225 for PHP `:8585` on the Windows guest — each labelled with what it does in the ranking. On Apache `:80` that reads *1 on the scored path, 14 starting a lower-scoring path, 71 starting no path the engine could trace* (Figure 6). The header states "the whole list, not a sample", because a truncated list that does not say so is the same failure as absent coverage. The panel is organised as three zones — what the weakness is, the two things a defender can *do* about it, and the evidence underneath — because an earlier version presented eight flat sections in which the fix ranked fourth, outweighed by a raw CVSS vector string and up to seventeen reference URLs. The CVSS vector is still quoted verbatim, but under *Evidence*; what leads is the same vector read back as a sentence (*"reachable from the network, needs no account, leads to full compromise"*), labelled as a reading of the vector rather than as a new claim. Both CVSS vocabularies are mapped, because 36.6% of the findings across the three hosts carry only a v2 vector and a v3-only reading would fall silent on a third of the list.

Bilingual, under one rule.. The interface is available in English and Arabic (396 interface strings, pinned at exact parity by test), under a rule that the chrome translates and the evidence never does: CVE identifiers, host and service names, CPE strings, scores and CISA's required-action text remain verbatim in every language.

A fourth host, scanned live through the interface.. The three evaluation hosts above are replayed from committed scans. To exercise the live path through the product rather than the command line, the dashboard was pointed at a containerised web service on the loopback address and the scan run from the page. It returned 22 assets of which **1** could be assessed, 101 findings, a 125-node / 177-edge graph, and 56 enumerated paths from that single entry point, the best of them reaching root at 59.4 — and the run line changed from "offline replay, no packets sent" to "LIVE SCAN — packets were sent to this target". Two things are worth reporting from it. The single assessed service was reached through the same scope gate as every other scan; and of the 21 assets it could not assess, two were APHE's own dashboard ports, which is a fair illustration of why coverage is reported rather than implied. That host is also the one on which the singular-case wording was first exercised — every committed fixture has more than one way in, so "1 way in" could not appear until a host with a single service was scanned. Every figure in this section is captured from the replayed evaluation host, so the screens and the numbers quoted throughout describe one and the same run — including Figure 8, the case in which the model *disagrees* with the ranking. An earlier revision had to reach for the loopback run to show that disagreement; it now occurs on the committed fixture, which makes the demonstration reproducible rather than anecdotal.

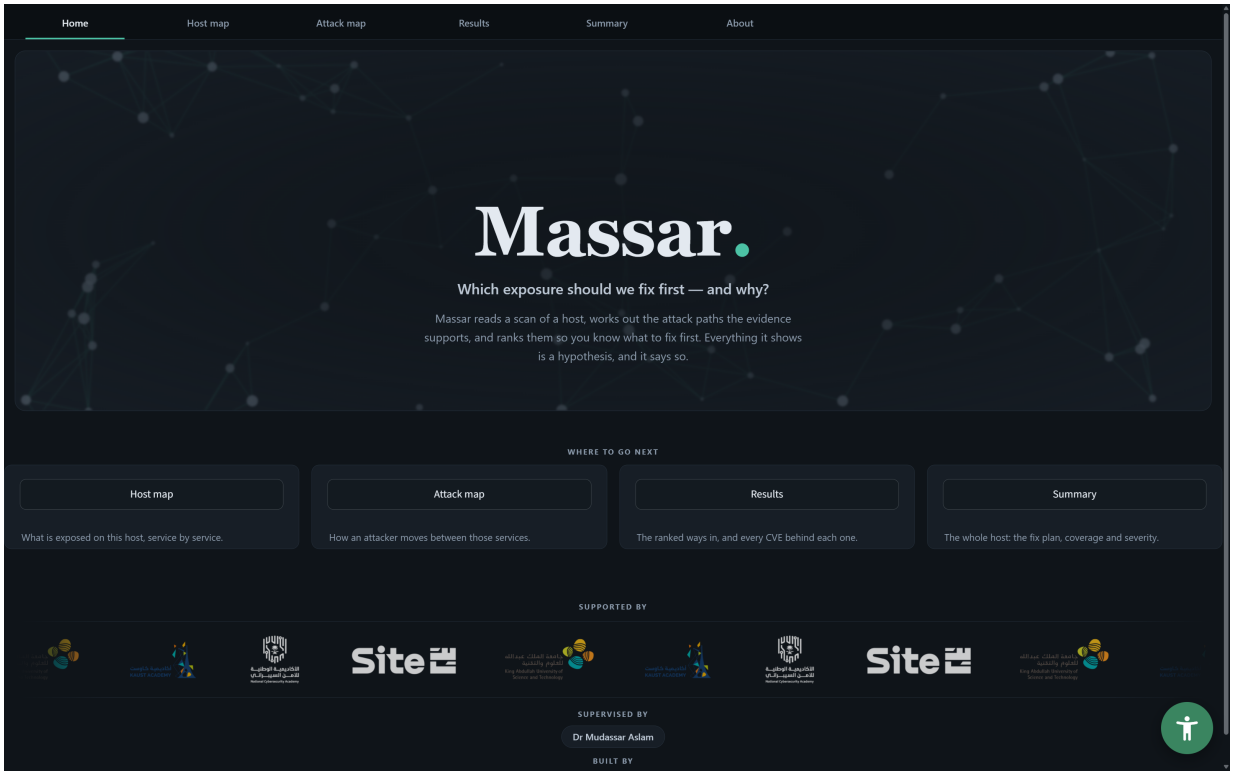


Figure 2 *Home* — the front door, and the one page that deliberately states no figure and loads no run, so that a failure in the pipeline can never break the way in. It is also where the product name a user sees, *Massar*, appears: the engine described throughout this report as APHE and the application in these screenshots are the same system. The four tiles hand off to the pages that do carry results.

4.3 Is the ranking any good? An honest evaluation

The engine's product is prioritisation, so it must be measured against the two trivial sorts a defender would otherwise use: rank-by-CVSS and KEV-first. We report the comparison whether or not it flatters the engine. On the real host the engine's order is *identical* to a KEV-first sort on all

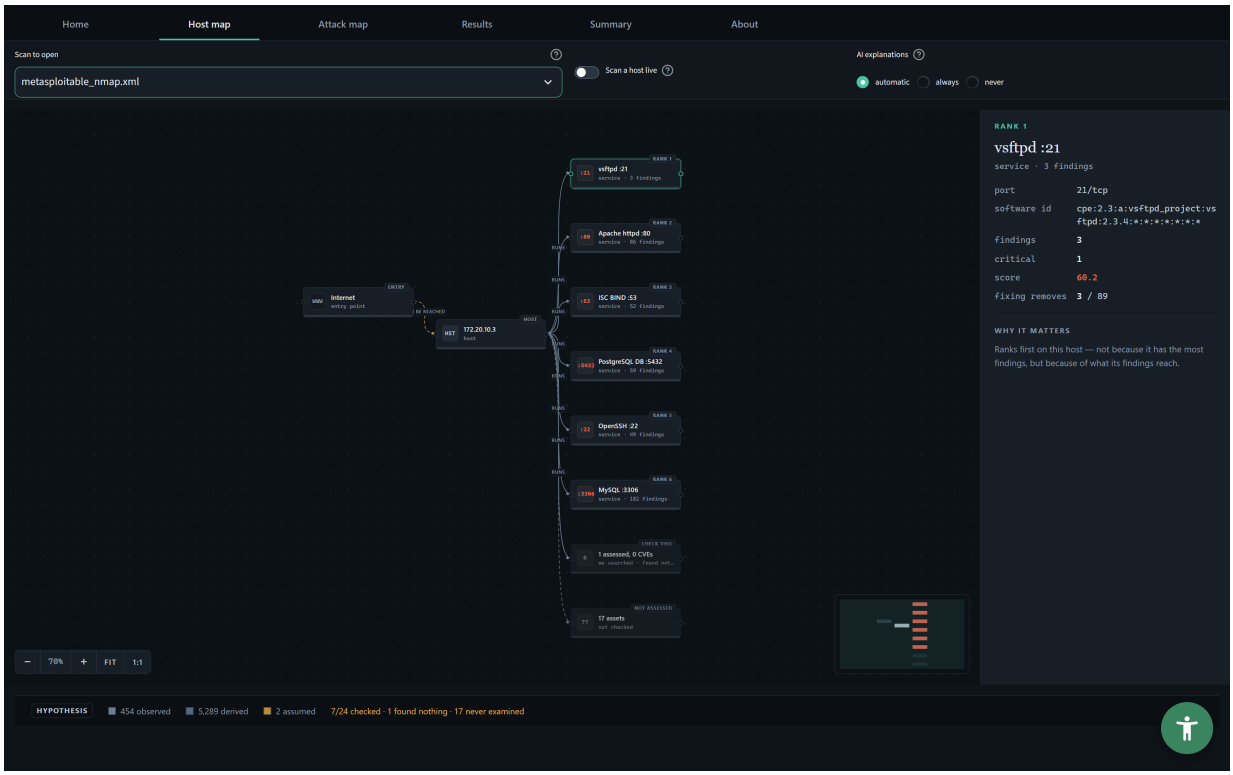


Figure 3 *Host map* — what is exposed on the evaluation host. The internet entry, the machine, and the six services that produced findings, each badged with its rank; the inspector shows the selected service's resolved CPE — vendor `vsftpd_project`, the NVD-verified spelling that recovers the backdoor CVE the scanner's own vendor string matches zero of — together with its findings, criticals and score. Two grey nodes are drawn rather than omitted: the sixteen services that could not be checked, and the seventh assessed service, which was looked up and returned nothing. The strip along the foot is permanent — the provenance census of the graph's edges (*454 observed, 5,289 derived, 2 assumed*) and the coverage split of the host's assets (*7 of 24 checked, 1 found nothing, 17 never examined*).

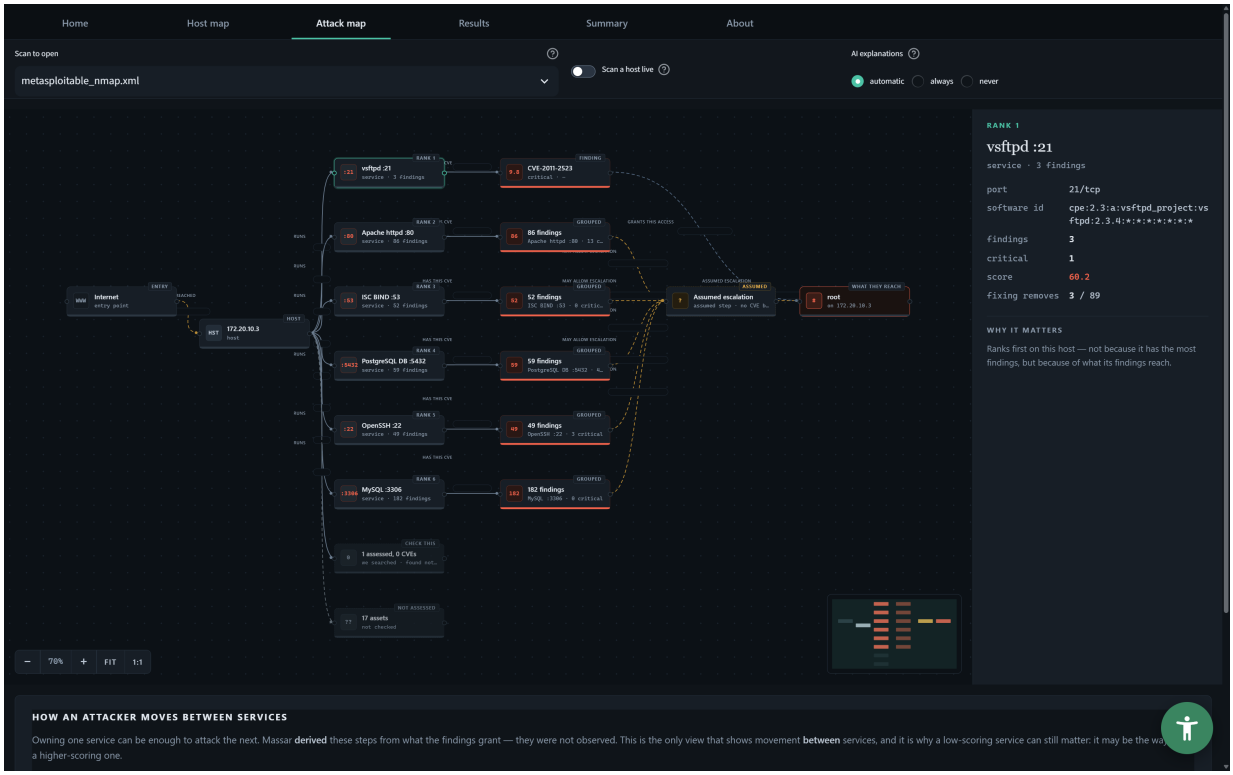


Figure 4 *Attack map* — how the exposure connects, end to end: internet → host → service → the grouped findings it carries → the `assumed` escalation rung → root. The assumed step is drawn in the warning hue and labelled ASSUMED wherever it appears, never folded into a service, because it is the one step in the chain that no CVE supports. The ranking is legible directly from the topology: `vsftpd :21` reaches root *without* passing through the assumed node, while the other five services must, which is why it ranks first on 3 findings against MySQL's 182.

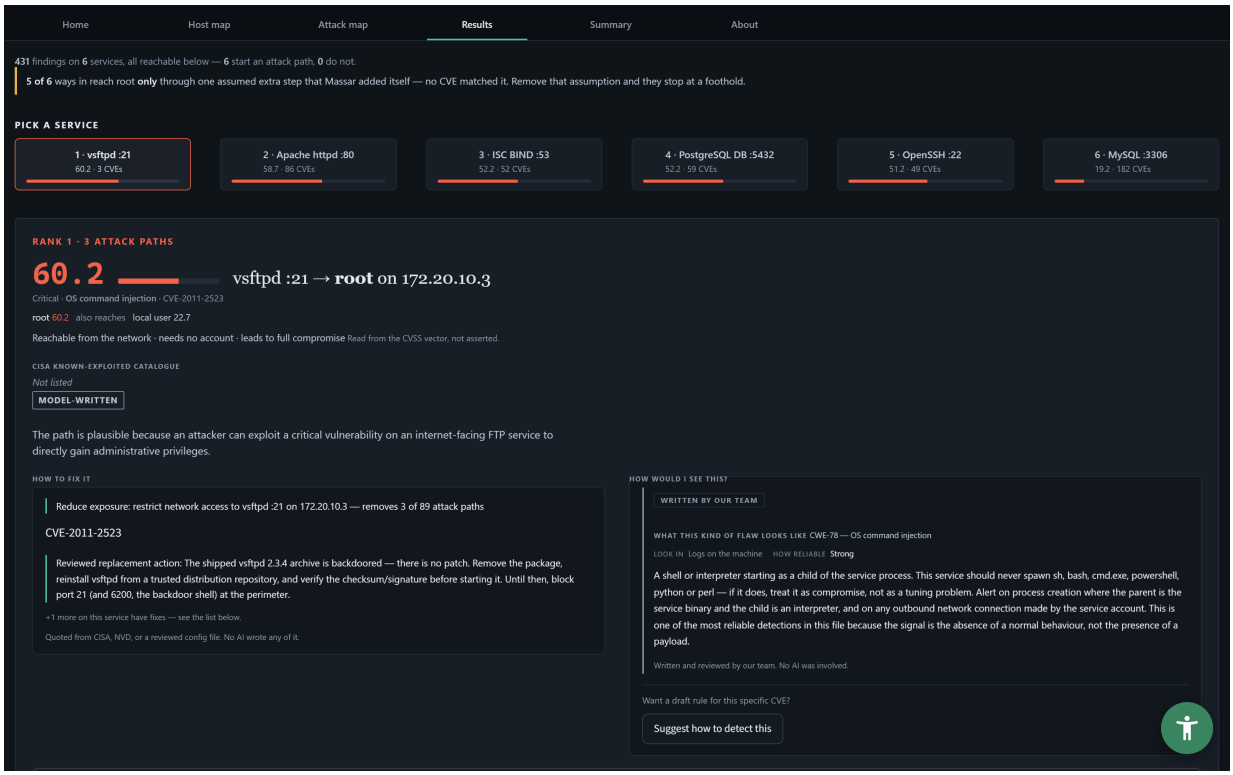


Figure 5 *Results* — the ranked ways in on the evaluation host, and the lead entry in full. The score, what it reaches, and *both* things a defender can do about it sit on one surface. The strip carries this paper's central claim without commentary: each entry is labelled with its score *and* its finding count, reading 60.2 · 3 CVEs at rank 1 and 19.2 · 182 CVEs at rank 6. The rationale is badged MODEL-WRITTEN. The two boxes below it are deliberately adjacent and have opposite provenance: *How to fix it* ends "no AI wrote any of it" and its *reviewed replacement action* is a curated, human-committed fact — the vsftpd 2.3.4 archive is backdoored, so there is no patch to apply — while *How would I see this?* carries class-level guidance written by the team and a button that asks the model for a rule specific to this CVE (Figure 9).

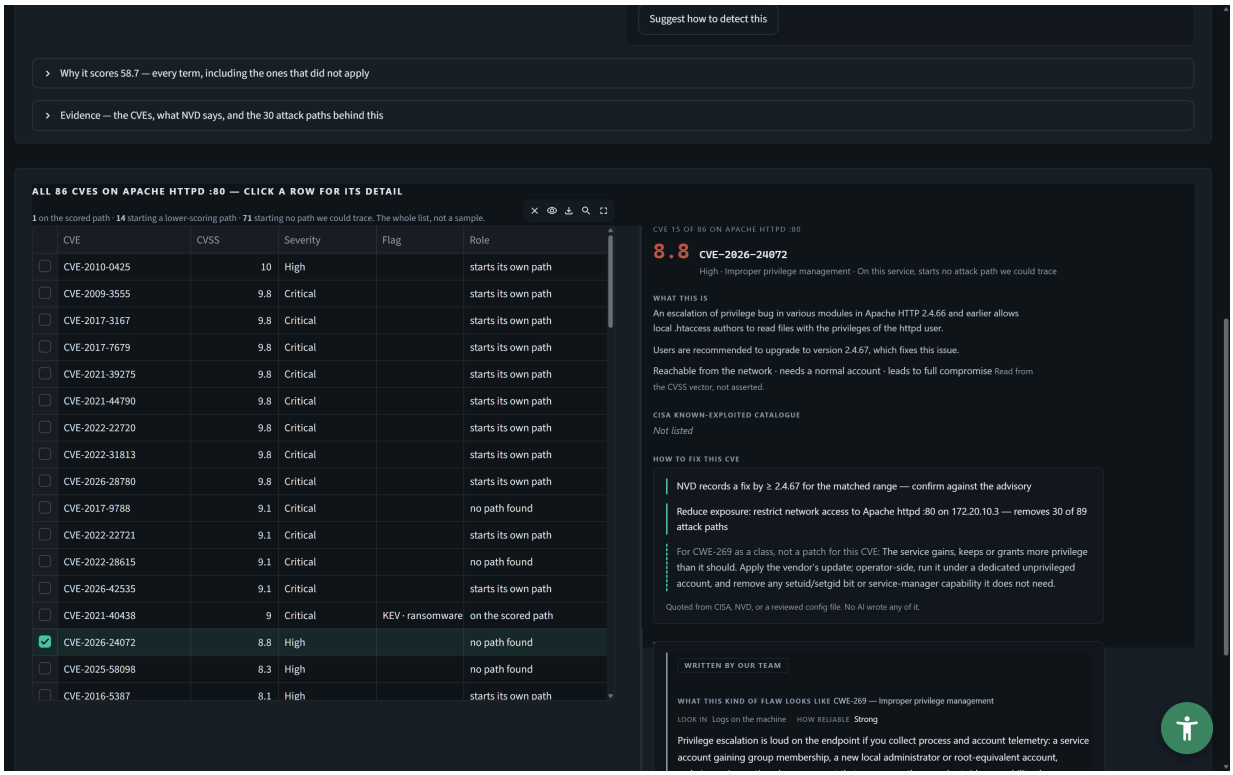


Figure 6 *Results*, lower half — all 86 CVEs on Apache `:80`, with a detail panel per row. The count line says what each finding does in the ranking — *1 on the scored path, 14 start a lower-scoring path, 71 start no path we could trace* — and states that this is the whole list rather than a sample. The panel reads in three zones: what the weakness is (NVD's description, then the CVSS vector rendered as a sentence and labelled as a reading of the vector rather than a new claim), what to do about it, and the evidence folded underneath. The fix box distinguishes what applies to *this CVE* from what applies to its *class*: the dotted entry is a CWE-level control, and says so in its own first words.

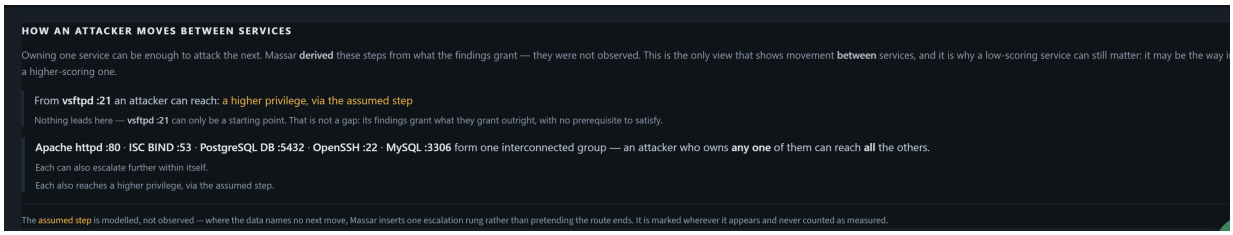


Figure 7 *Attack map*, lower half — service-to-service escalation as sentences rather than as a hairball or a matrix of counts. It states which routes *exist* and marks the assumed step, and on this host it reports two facts no other view shows: `vsftpd :21` is a service *nothing leads to* — it can only be a starting point, which is not a gap but the reason it ranks first — while the remaining five form one interconnected group in which owning any member reaches all the others.

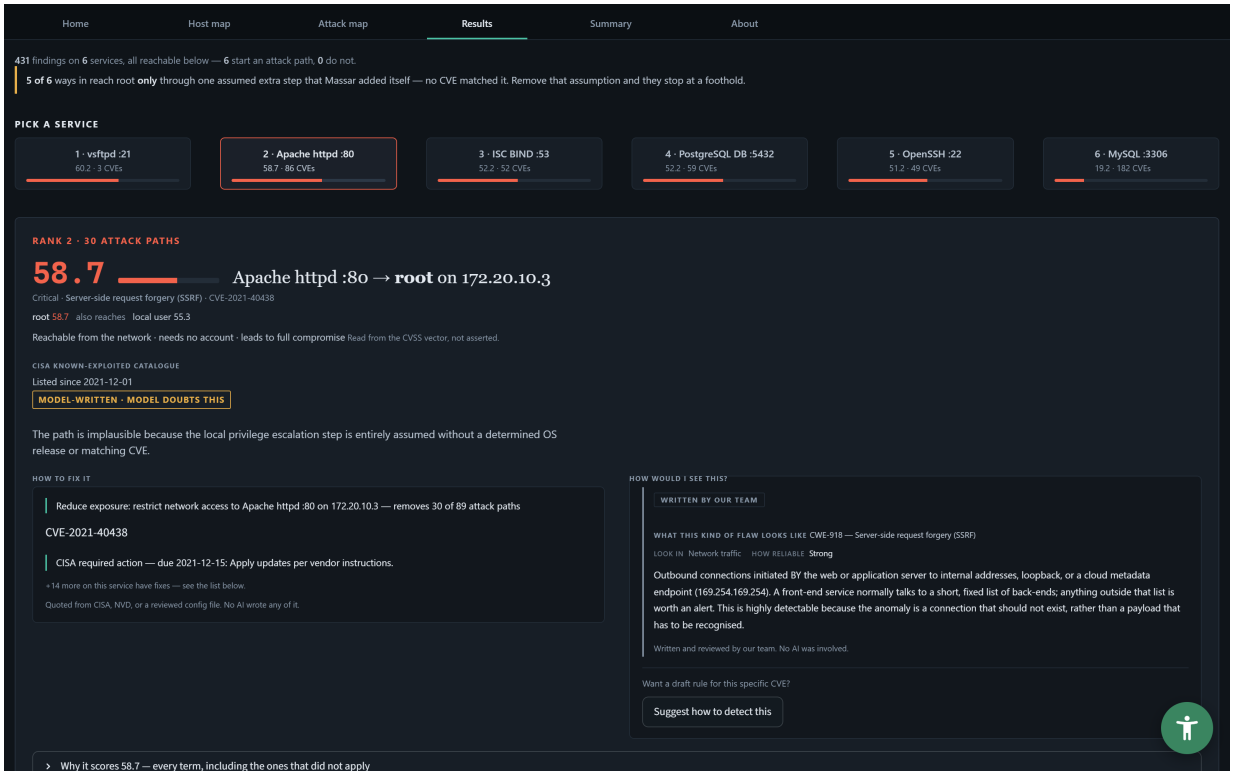


Figure 8 The separation of computation from judgement, visible in the product — and on the evaluation host itself. The model judged the second-ranked path *implausible*, and said why: *"the local privilege escalation step is entirely assumed without a determined OS release or matching CVE"* — which is a fair criticism of the assumed-escalation rung of §4, and the engine ranked the path at 58.7 regardless. The verdict is badged MODEL DOUBTS THIS beside MODEL-WRITTEN, shown to the reader, and changes no number. A model permitted to disagree in public, and still unable to move a rank, demonstrates the guarantee of §4.3 more convincingly than one that always agrees.

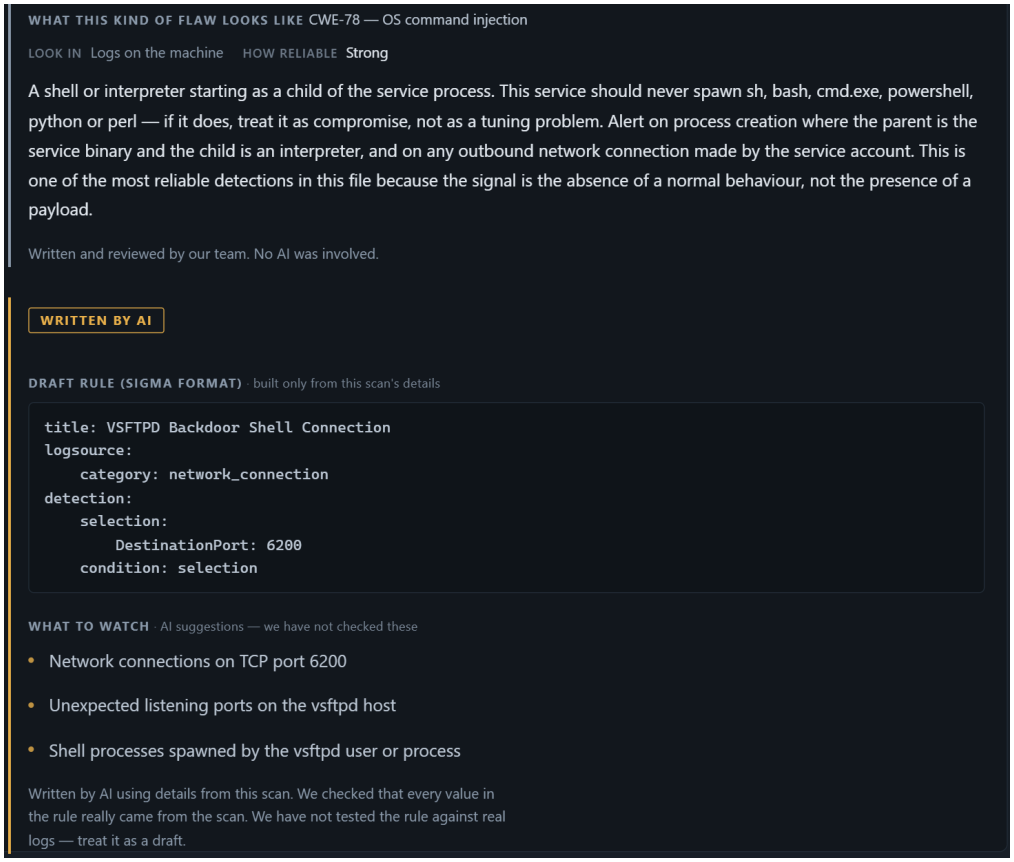


Figure 9 The two detection rungs on the top-ranked finding, and their two different claims. Above: class-level guidance for CWE-78, written and reviewed by the team, stating where to look (*logs on the machine*) and how reliable the signal is (*strong*) — available with no provider configured. Below: the model's per-CVE draft, admitted by the guard because its one concrete indicator, destination port 6200, is quoted in NVD's own description of the vsftpd 2.3.4 backdoor rather than invented. The two footers make the difference in provenance explicit, and the lower one refuses the stronger claim in its own words: the free-text watch items are *"AI suggestions — we have not checked these"*, and the rule itself has not been tested against real logs.

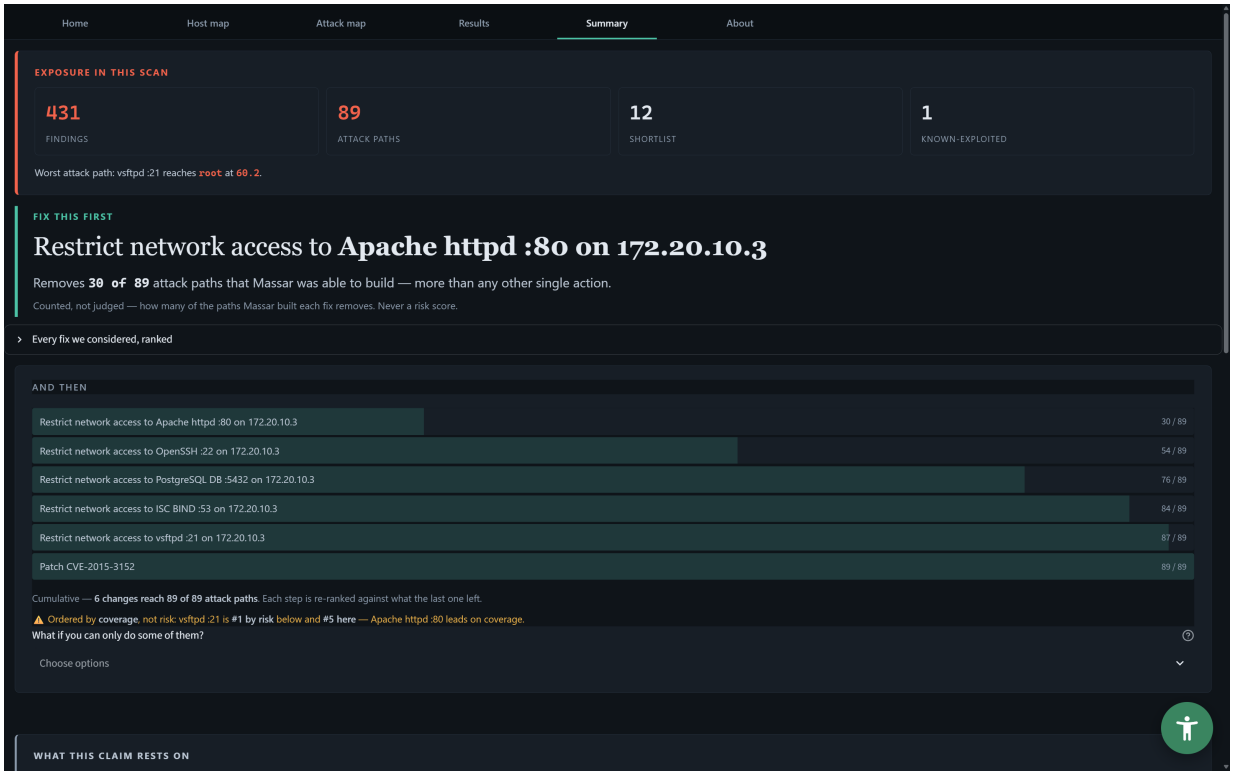


Figure 10 *Summary* — the whole host at once, and the page on which the two orderings are made to disagree in public. The greedy plan is shown as a residual: each bar is what that change removes given everything above it, running 30 → 54 → 76 → 84 → 87 → 89, so a reader sees that three changes clear 85% of the graph rather than only that the best single change clears 34%. The warning line beneath states the disagreement outright — *ordered by coverage, not risk: vsftpd :21 is #1 by risk below and #5 here* — rather than leaving a reader to assume the two lists agree.

but 89 of 3,916 ranked pairs (2.3%); on the small uniform-severity fixtures it is identical (0%). This is a property of the data, not a defect of the design: the score is a product of CVSS, KEV, exposure, chain-length, and privilege terms, and when a host's exposed CVEs are uniformly critical and known-exploited the first four are constant, leaving only the privilege term to differentiate — and that term acts only when chains reach different privilege levels. Where they do, the difference is decisive rather than marginal: in the demonstration case both candidate chains carry the same maximal CVSS and KEV status, so a severity sort cannot separate them, yet the engine correctly ranks the chain that reaches root above the one that stops at a foothold. Across the three labelled scenarios the engine's top-ranked path matches the human-labelled correct answer in all three. We are deliberately precise about the source of that improvement: even with the assumed-escalation node removed the order still differs from KEV-first (5 of 990 pairs), but that residual difference rests *entirely on one confirmed direct-root backdoor* (CVE-2011-2523), not on the exploitability term alone — the differentiator has moved from a modelling *assumption* to a single curated *fact*. We report the measured 2.3% as the honest statement of the engine's marginal value on this target: firmer footing than an assumption, but still one data point, not an unconditional improvement over severity ranking.

A metric on which the engine loses.. Rank agreement depends on human labels, so we also measured a label-free one: *chokepoint efficiency* — fixing in the engine's order versus in CVSS order, and asking how much of the enumerated attack surface each successive fix removes. On the real host the engine is *worse*: an area-under-curve of 0.509 against 0.563 for a plain CVSS ordering, with both needing the same six actions to cut every chain; on the two smaller fixtures the two orders tie exactly. We report this because it is the honest result and because it is informative rather than merely unflattering: the two orderings optimise different things. Chokepoint efficiency rewards fixing whichever node participates in the most enumerated chains, which on this host is the Apache service carrying the largest CVE count; the engine instead ranks by the severity of the *outcome* an attacker obtains, which puts a single confirmed instant-root backdoor first even though it appears in only two chains.

This is not a distinction we invented to explain a loss. Stergiopoulos et al. [16] compute both orderings over the same attack graphs — a risk ranking over attack paths, and a centrality-based ranking of individual nodes — and report them side by side as separate results; across both of their case studies the two lists share no entries, and they conclude that some vulnerabilities matter for securing specific end services while others matter for securing a network against as many attacks as possible. Severity-optimal and coverage-optimal fix orders are therefore known to diverge, and a ranking built for one objective should not be judged by the other's metric.

APHE ships both answers rather than choosing: the worklist orders by outcome severity, and `graph/cuts.py` answers the coverage question separately as the "highest-coverage fix". Scored on this metric that component reaches an AUC of 0.703 against the CVSS sort's 0.563. **We do not offer that number as a rebuttal.** `cuts.py` greedily maximises chains-cut-per-fix, which is precisely the quantity chokepoint AUC integrates, so its lead is close to definitional: it demonstrates only that the component built to answer the coverage question does answer it. The figure that tests the contribution of this work is still the worklist's 0.509, and on that figure the engine loses to a plain severity sort. We report it because reporting only the arm that flatters us would be the larger failure.

One difference is worth stating precisely, because it cuts the other way. The prior work above infers a fix's value from a node's centrality, without re-enumerating the graph after that fix to confirm how much attack surface actually disappeared; `cuts.py` reports a directly measured count — for example, restricting Apache on port 80 removes 30 of the 89 enumerated chains.

Measuring the quantity rather than proxying it is a modest advantage, and it is independent of which ordering a defender ultimately follows.

The language model changes explanation, not rank.. The separation of computation from judgement is enforced by an ablation test that runs the scorer three ways — with no model, with a model that calls every candidate plausible, and with one that calls every candidate implausible — and asserts that the ranked paths and their scores are byte-identical across all three; only the free-text rationale differs. A live round-trip against Google Gemini confirms the model's structured output is correctly grounded in the graph's privilege labels and is reproducible at temperature zero. This is what lets us claim, rather than merely assert, that the ranking is deterministic.

4.4 Verification and component status

Test suite.. The automated suite collects 904 tests, of which 900 pass and 4 are expected failures (discussed below); it runs in roughly a minute and is concentrated on the risky logic. The largest groups are 319 for the dashboard's presentation layer (`test_dashboard_view`), 82 for the detection-guidance layer and its guard (`test_detect`), 80 for the scope gate (`test_scope`), 48 for vulnerability matching (`test_vulns`), 47 for the scan wiring (`test_gather_wiring`), 38 for the language-model agents stage (`test_agents`), 35 for the `nmap` ingest and CPE handling (`test_nmap_ingest`), 27 for the investigation canvas (`test_dashboard_canvas`), 22 for deterministic remediation (`test_remediation`), 20 for the audit fixtures (`test_audit_fixtures`), 19 for worklist grouping (`test_prioritize`), 18 each for the shared model-call runtime and the ranking evaluation (`test_eval`), 17 for CPE enrichment (`test_enrich`), 15 for scoring (including the LLM-invariance ablation and the exploitability weighting), and 14 each for graph construction and the highest-coverage-fix cuts, alongside 71 tests across nine smaller suites covering the typed contracts, the assumed-escalation node, privileges, the end-to-end pipeline, coverage, direct-root labelling, multi-host enrichment, and the real-scan replay. Static linting with the project's configured `ruff` rule set reports no issues.

The four expected failures are deliberate and are the most useful tests in the suite. Each encodes a property the design *specifies* but the engine does not yet satisfy — most pointedly, that no chain may traverse a host-to-host step without a foothold already held on the source host. They are marked strict, so they fail the build the moment the engine begins to satisfy them, which turns a known gap into something that reports itself rather than something that has to be remembered. All four came from a team member auditing the design document against the code; reproducing one of them independently then exposed an ambiguity in the design document itself rather than in the implementation — the clause it relies on names a CVE reachable as a first step, which is not the same as a CVE that grants a position to move from.

The presentation layer carries the largest share because the interface makes claims that the engine cannot check for itself. Those tests pin, among others, that no synthesised graph edge is ever drawn as *observed*; that the two hedged relation labels keep their qualifier in both languages; that the Arabic and English string tables are at exact key parity; and that the message shown when a run fails names a control the interface actually offers. Three of them exist because the property they pin had already been violated in a shipped build without a single test turning red.

Test *count* is not test *quality*, and we have repeatedly caught our own assertions being vacuous — passing regardless of whether the behaviour they named was present. One case is instructive: a test asserting that raw service banners never reach the language-model provider set the banner on the service record, but the graph builder never copies that field onto a node, so the string could not have reached the prompt whatever the sanitiser did. It pinned a different module's field selection. Because this recurred across several work sessions, it is now a standing practice rather

than an observation: *every* new assertion is verified by deliberately introducing the defect it claims to catch and confirming the test turns red. That practice has since caught a test that counted items rather than checking them and stayed green when one was removed; one that returned the expected value for an unrelated reason, so it passed with the guard it existed to test disabled; and a branch that turned out to be unreachable as behaviour, which we deleted rather than tested around. We report this because a passing suite is evidence only if its assertions can fail.

Component status.. Table 3 records the maturity of each stage honestly. Every deterministic stage is implemented and tested; the language-model layer is integrated and its non-influence on rank is proved; the parts that remain partial are named as such.

Table 3 Stage maturity. Detailed in Section 6.

Stage	Status
gather	real `nmap` ingest, single- and multi-host; scan driven from the pipeline, scope-gated; Shodan not built
enrich	CPE via scanner identifiers → curated table → NVD dictionary, plus banner-named and misidentified products
vulns	implemented + tested — version-range matching, NVD + KEV, live feeds or recorded sidecar
graph	implemented + tested — chaining, assumed-escalation, edge and enumeration budgets
hypothesis	deterministic enumeration + integrated LLM plausibility layer (rank-invariant)
scoring	implemented + tested — explainable breakdown, worklist grouping
remediation	implemented + tested — copied verbatim from CISA/NVD, containment computed from the graph; no model text
detection	implemented + tested — curated class rung (62.4% of findings) + guarded model draft; a hypothesis, not a validated detection
dashboard	implemented + tested — six pages, bilingual, live scan run from the page and savable for offline re-opening

5 Discussion

The ranking in Table 1 is the clearest evidence that APHE measures attacker *outcome*, not isolated severity. The root chain scores above the foothold even though its added CVE (Dirty Pipe, CVSS 7.8) is individually less severe than Log4Shell (10.0): the score folds the privilege *gained* into impact, so reaching operating-system administrator outranks stopping at a service account. A CVSS-sorted list would invert this — ranking the 10.0 foothold above the 7.8 escalation — and hide the fact that, together, the two yield root. Encoding the attacker's end state is what makes the output actionable.

The result also shows why the graph must distinguish *where* a vulnerability lives. Because Dirty Pipe is a kernel flaw with no network port, it attaches to the host (`has_local_vuln`) rather than to a service, and the entry-admissibility rule then refuses any path that reaches it directly from the internet. This one modelling decision — that a vulnerability's reachability depends on whether it is network-borne or host-local — is what separates a realistic attack graph from a flat list, and it is enforced deterministically, so the guarantee does not depend on a model behaving

well.

That deterministic foundation is a deliberate hedge. Prior work is consistent that a language model asked to drive a multistage attack at the level of individual low-level actions is unreliable, and that raising the abstraction it plans over helps substantially: Incalmo plans in high-level declarative tasks executed by domain-specific agents and acquires critical assets in 37 of 40 emulated multi-host networks, against 3 of 40 for existing LLM-assisted offensive systems [9], while PentestGPT identifies loss of long-horizon context as its dominant failure mode [8]. APHE takes the strong form of that lesson: because the deterministic core already produces a correct, explained ranking, the integrated language-model layer is an *enhancement* to plausibility judgement rather than a load-bearing component.

That containment came at a price we had to measure rather than assume. Diffing a full run with the model enabled against the same run with it disabled changes *seven lines* of output on the evaluation host: the line recording which mode the run used, and six rationale sentences. Findings, graph, ranking, scores and remediation are byte-identical — and among those six the model calls the top-ranked path plausible and four of the assumed-escalation paths implausible, without moving any of them. That is the ablation guarantee holding in a live run rather than in a test. But a layer that safe is also, on that evidence, close to decorative — which is why the model was given detection guidance as a second job (Section 3.4). The design constraint is the same one in both cases, and it is the transferable point: give the model work for which no authoritative source already exists, and admit its output only through a deterministic check. Plausibility judgement satisfies that because the candidate set is enumerated by code and every verdict is matched back to a real path; detection guidance satisfies it because a guard rejects any indicator not traceable to the scan. Remediation satisfies neither test, and so the model is excluded from it. The main risk this leaves is not correctness but *coverage*: the privilege postcondition is an honestly-labelled *upper bound* derived from public CVSS and CPE data, which can over-attribute an application flaw that happens to grant root. Whether an upper bound is acceptable for a defensive prioritiser, or should be tightened by per-CVE curation, is the project's main open question (Section 6).

6 Limitations and Future Work

The system is an honest partial build. Its limitations are stated plainly below, each paired with a concrete next step.

- **Marginal ranking benefit on uniform hosts.** On a host whose exposed CVEs are uniformly high-severity and known-exploited, the ranking coincides with a KEV-first sort (Section 4); the engine's measured differentiation on the real target was 2.3% of ranked pairs. The privilege-and-chaining dimension — the one thing a severity list omits — is the only differentiator, and it acts only when chains reach different privilege levels. *Next:* extend the chaining model beyond privilege to a knowledge/credential axis (an information-disclosure that leaks credentials enabling an authenticated flaw), which CVSS confidentiality impact makes derivable from public data; and evaluate on a wider set of real hosts.
- **A first-time host is a single command, but it needs the network.** Committed results are key-free: the NVD and KEV responses for each evaluation host's assessed CPEs are stored as a sidecar, so every reported figure reproduces offline on a fresh checkout. A host that has never been recorded is now also a single command — the pipeline accepts a target, scans it through the same scope gate, and queries the live NVD and KEV feeds directly (measured end to end on a previously unseen host in 51 seconds; 278 seconds for a full-port scan of the Windows guest, where the cost is the scan profile rather than the engine). The residual

limitation is narrower than it was: that live path needs reachable feeds and, without an API key, is throttled to five requests per thirty seconds. *Next:* cache the live responses into a sidecar automatically, so the second run of a new host is offline without a separate recording step.

- **Escalation on an unversioned OS is a modelled assumption.** When the scanner identifies the operating system but not its exact release — as on the real host, where the kernel is reported only as a version series — the engine cannot name a kernel CVE, so it inserts a single, clearly-labelled assumed local-escalation node rather than assessing the thousands of unconfirmed CVEs the coarse identifier would otherwise match. This restores the root-reaching chains the data supports, at the cost of one explicit assumption; it is never counted as a finding and carries `assumed` provenance throughout. *Next:* narrow it as OS identification improves.
- **Enrichment still falls back to a curated table.** CPE resolution now prefers the scanner's own identifiers and the NVD dictionary, but products those miss fall to a small hand-entered table and are otherwise reported as unidentified. *Next:* a model-assisted matcher for the residue, validated against NVD before use, remains a stretch goal.
- **Vulnerability lookup depends on external feeds.** The live NVD client (paginated) and the CISA KEV client are implemented and cached to disk. The committed fixtures now cover the demonstration CVEs *and* the real host's seven assessed CPEs verbatim, so the offline path exercises the matching logic against real catalogue data; but any host outside those recordings still depends on the live feeds being reachable, and a transport failure part-way through a run currently aborts it rather than returning a partial result with a stated gap. *Next:* degrade gracefully on feed errors and report the gap.
- **Host-to-host reachability is never measured.** Multiple hosts are parsed, enriched, graphed and ranked together — and three real machines have now been evaluated, on two operating-system families (Metasploitable 2, and both the Ubuntu and Windows Server guests of Metasploitable 3) — but every host is modelled as reached directly from the internet entry point. The graph builder accepts a measured topology and stamps such edges `measured`, yet nothing in the pipeline produces one, so APHE never claims a pivot from one host to another. *Next:* feed connectivity measured between authorized laboratory hosts, and gate multi-host chaining on a foothold existing before any lateral step.
- **No operating-system CVE is assessed on any real host, and this is not a limitation a better scanner would remove.** Deciding whether a scanner has named a real release or only a release series is done by counting dotted version segments, and at least three are required. The guard exists for a sound reason: the series `linux_kernel:2.6` matches several thousand CVEs and, before it, produced over 1.3 million graph edges and non-terminating path enumeration. But a segment count is a Linux-kernel-shaped rule applied to every operating system, and NVD's own CPE dictionary shows it cannot be tuned into correctness. CentOS has *zero* three-segment entries — `centos:6.5` is as precise as CentOS is ever written — and every Windows Server 2008 entry carries a version of `r2` or `-`, with release identity split across the `product`, `version` and `update` attributes and no dot anywhere. Ubuntu publishes both `12.04` and `12.04.5` as real releases, so even a per-product segment count is ill-defined. Of the 1,219 distinct operating-system CPEs in `nmap`'s fingerprint database, 282 (23.1%) pass the test.
  This includes our own evaluation host. Metasploitable 2 is fingerprinted as `cpe:/o:linux:linux_kernel:2.6` — a series — so it is binned version-undetermined like the rest, and all seven assessed CPEs on that host are applications. Every one of the 431 reported findings is

a service CVE.
Service-level assessment is unaffected on every platform tested — Apache, OpenSSH, MySQL and Microsoft IIS all resolve and are assessed normally — and a host whose operating system is identified but unversioned still receives the assumed local-escalation node, so chains still reach root. Verified on a synthetic Windows Server host: the service CVE is assessed and both a foothold chain and a root-reaching chain are produced. What is lost is the operating-system CVE layer alone.
Relaxing the guard would not recover it. Because 87% of the matched entries in our recorded NVD data express their vulnerable set as a wildcard CPE plus a version *range* rather than an exact version, a more precise version string does not reliably narrow the candidate set: measured against the NVD API, pinning `linux_kernel:4.14.296` instead of `4.14` reduces matches by only 12%, while `http_server:2.4.49` returns *more* than `2.4`, because a version string that is not itself a shipped release behaves as an exact-match miss rather than as a wildcard. The feeds that do carry per-release precision — OVAL and vendor security data — are *package-state* feeds whose tests compare installed package versions, which requires authenticated or local host access rather than a network banner. We therefore state the limitation in its strongest honest form: operating-system CVE assessment at release granularity is not achievable from unauthenticated network scanning and NVD alone. *Next:* replace the segment count with a direct blast-radius control — assess an identified operating system only while its candidate CVE set stays below a budget, and otherwise fall back to the labelled assumed-escalation node the pipeline already builds.

- **Enumeration is bounded, but truncation is not guaranteed to be harmless.** Scale has been a first-class concern for practical attack-graph tooling since at least [17]. The quadratic step that pairs same-host weaknesses into chaining edges is capped at 50,000 edges; on the real host it produces 5,289 and the cap is not reached. That cap alone proved insufficient, and the reason is worth stating because it was a design error rather than a tuning one: it counted *edges*, while the quantity that actually grows is *paths*. Measured on a synthetic privilege ladder, 80 findings produced 168,420 paths in 124 seconds while the edge cap sat at 2% of its limit and could never have fired in time. Enumeration therefore carries its own budgets — 500,000 edge traversals and 50,000 paths per endpoint, both count-based so a run stays reproducible — and a single traversal replaced one per endpoint, taking the same synthetic case to 2.0 seconds and the real host's whole run to well under a second. Metasploitable 2 spends 10,963 of its 500,000 traversals. *The residual limitation is honest labelling, not absence:* when a budget is reached the run says so, but candidates are generated in traversal order rather than score order, so a dropped chain is not guaranteed to be a low-scoring one. *Next:* generate candidates in an order that lets truncation be safe as well as declared.
- **Privilege labels are an upper bound.** The postcondition is derived from CVSS and CPE and can over-attribute privilege to an application flaw. A curated allowlist now corrects this in the opposite direction for confirmed runs-as-root flaws (Section 4), but it is a small, manually reviewed file rather than a general solution, and the underlying label remains an upper bound. *Next:* broaden the curated set and obtain guidance on whether an honestly-labelled upper bound is acceptable for a defensive tool.
- **Scoring weights are unvalidated.** The likelihood-times-impact weights are a defensible configuration choice, not an empirically-tuned model, and EPSS is not yet used. *Next:* incorporate EPSS [14] as a likelihood signal and evaluate the weighting against labelled scenarios.
- **The detection guidance is a hypothesis, and its usefulness is unmeasured.** The guard

establishes that every concrete indicator inside a proposed rule traces back to the scan's evidence (Section 3.4). That is a claim about *grounding*, and it is the only claim we make. It does not establish that a rule fires on real traffic, that it is specific enough not to bury an analyst in false positives, or that the telemetry it assumes is being collected at all — and it says nothing about the free-text watch items beside the rule, which are shown as unchecked suggestions because the guard does not check prose. Structural validity was measured at 93.3% on a stratified sample; usefulness was not measured, and the two must never be quoted as one number. Yield is low by construction: only about three percent of findings produce an accepted rule, and 234 of the 534 class-level entries that fire are labelled a weak signal. *Next:* the missing measurement is a human one. A 24-item sheet, deliberately stratified *against* the feature so that guard refusals and no-guidance cases are over-represented, is prepared and awaiting rating by a detection engineer — a role distinct from the offensive-security experience on this team, which is why we have not rated it ourselves. Beyond that: emit accepted rules as files a security-operations platform can ingest, and add a second rule format only together with a validator the guard can apply to it.

7 Conclusion

APHE is a working pipeline that turns a host's exposed services and known vulnerabilities into a ranked, grouped, and explained set of attacker paths with attached remediation. It runs offline on the demonstration host — producing the two-CVE privilege-escalation chain and correctly ranking the root chain (65.3) above the foothold alone (61.4) — and has additionally been run end-to-end against three live hosts on two operating-system families, the largest yielding 431 findings ranked end-to-end in 0.7 seconds from the recorded scan, backed by 900 passing tests. A deterministic core owns the attack graph and the score, so the ranking is reproducible and defensible; the integrated language model is confined to a bounded, grounded plausibility judgement whose non-influence on rank we prove by ablation, and to drafting the detection guidance a deterministic guard admits only when every concrete indicator it names traces back to the scan's own evidence. It never writes remediation, which is copied verbatim from the feeds. The real run is also what keeps the contribution claim honest: where a host's exposure is uniformly severe the ranking coincides with a KEV-first sort, and the engine earns its place through the chaining dimension a severity list omits — measured, on that target, at 2.3% of ranked pairs, and decisive in the root-versus-foothold case severity alone cannot resolve. We also report the axis on which it loses: ordered by how fast successive fixes shrink the attack graph, a plain CVSS sort is more efficient, scoring 0.563 against our 0.509, because the engine optimises for the severity of the attacker's outcome rather than for graph coverage — a distinction the dashboard keeps visible by presenting the highest-coverage fix separately from the ranking. The value, then, is not an unconditional improvement over severity ranking but the surrounding judgement a flat list withholds: which weaknesses combine, which to fix first where the structure allows a distinction, how sure the tool is, and what to do about it — delivered by deterministic, auditable code, with the model kept to the one sub-task it is suited to.

References

[1] Oleg Sheyner et al. "Automated Generation and Analysis of Attack Graphs". In: *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P)*. 2002, pp. 273–284. DOI: 10.1109/SECPRI.2002.1004377.

[2] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. "MulVAL: A Logic-based Network Security Analyzer". In: *14th USENIX Security Symposium*. 2005. URL: https://www.usenix.org/legacy/event/sec05/tech/full_papers/ou/ou.pdf.

[3] Kerem Kaynar. "A Taxonomy for Attack Graph Generation and Usage in Network Security". In: *Journal of Information Security and Applications* 29 (2016), pp. 27–56. DOI: 10.1016/j.jisa.2016.02.001.

[4] M. Ugur Aksu et al. "Automated Generation of Attack Graphs Using NVD". In: *Proceedings of the 8th ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2018, pp. 135–142. DOI: 10.1145/3176258.3176339.

[5] Zakir Durumeric et al. "A Search Engine Backed by Internet-Wide Scanning". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015, pp. 542–553. DOI: 10.1145/2810103.2813703.

[6] Pratyusa K. Manadhata and Jeannette M. Wing. "An Attack Surface Metric". In: *IEEE Transactions on Software Engineering* 37.3 (2011), pp. 371–386. DOI: 10.1109/TSE.2010.60.

[7] M. Poisson et al. "CVE Representation to Build Attack Positions Graphs". In: *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*. arXiv:2312.02585. 2023. URL: https://arxiv.org/abs/2312.02585.

[8] Gelei Deng et al. "PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing". In: *33rd USENIX Security Symposium*. arXiv:2308.06782, preprint titled differently. 2024. URL: https://arxiv.org/abs/2308.06782.

[9] Brian Singer et al. *Incalmo: An Autonomous LLM-assisted System for Red Teaming Multi-Host Networks*. 2025. arXiv: 2501.16466 [cs.CR]. URL: https://arxiv.org/abs/2501.16466.

[10] Renascence Tarafder Prapty, Ashish Kundu, and Arun Iyengar. *Using Retriever Augmented Large Language Models for Attack Graph Generation*. 2024. arXiv: 2408.05855 [cs.CR]. URL: https://arxiv.org/abs/2408.05855.

[11] Andreas Happe and Jürgen Cito. "Getting pwn'd by AI: Penetration Testing with Large Language Models". In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 2023, pp. 2082–2086. DOI: 10.1145/3611643.3613083.

[12] Lingyu Wang et al. "An Attack Graph-Based Probabilistic Security Metric". In: *Data and Applications Security XXII (DBSec)*. Vol. 5094. LNCS. 2008, pp. 283–296. DOI: 10.1007/978-3-540-70567-3_22.

[13] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. "Dynamic Security Risk Management Using Bayesian Attack Graphs". In: *IEEE Transactions on Dependable and Secure Computing* 9.1 (2012), pp. 61–74. DOI: 10.1109/TDSC.2011.34.

[14] Jay Jacobs et al. "Exploit Prediction Scoring System (EPSS)". In: *Digital Threats: Research and Practice* 2.3 (2021). arXiv:1908.04856. DOI: 10.1145/3436242.

[15] Steven Noel et al. "Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs". In: *19th Annual Computer Security Applications Conference (ACSAC)*. 2003, pp. 86–95. DOI: 10.1109/CSAC.2003.1254313.

[16] George Stergiopoulos, Panagiotis Dedousis, and Dimitris Gritzalis. "Automatic Analysis of Attack Graphs for Risk Mitigation and Prioritization on Large-Scale and Complex Networks in Industry 4.0". In: *International Journal of Information Security* 21.1 (2021), pp. 37–59. DOI: 10.1007/s10207-020-00533-4.

[17] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. “Practical Attack Graph Generation for Network Defense”. In: *22nd Annual Computer Security Applications Conference (ACSAC)*. 2006, pp. 121–130. DOI: 10.1109/ACSAC.2006.39.