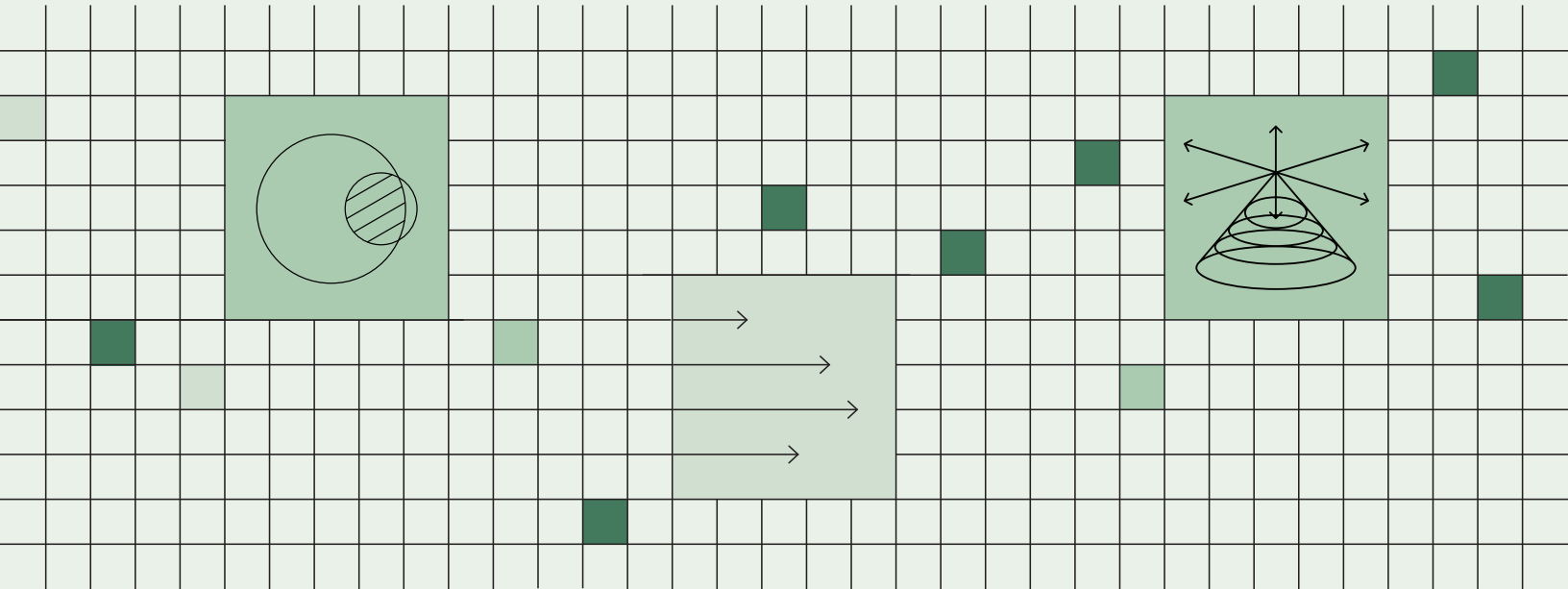
EMPIRICAL

Behind The Empirical Security Global Model



Contents

3 Introduction

4 How the Empirical Global Model Supports Better Vulnerability Management

6 Model Inputs: What We Measure

8 How The Models Learn

9 Validation

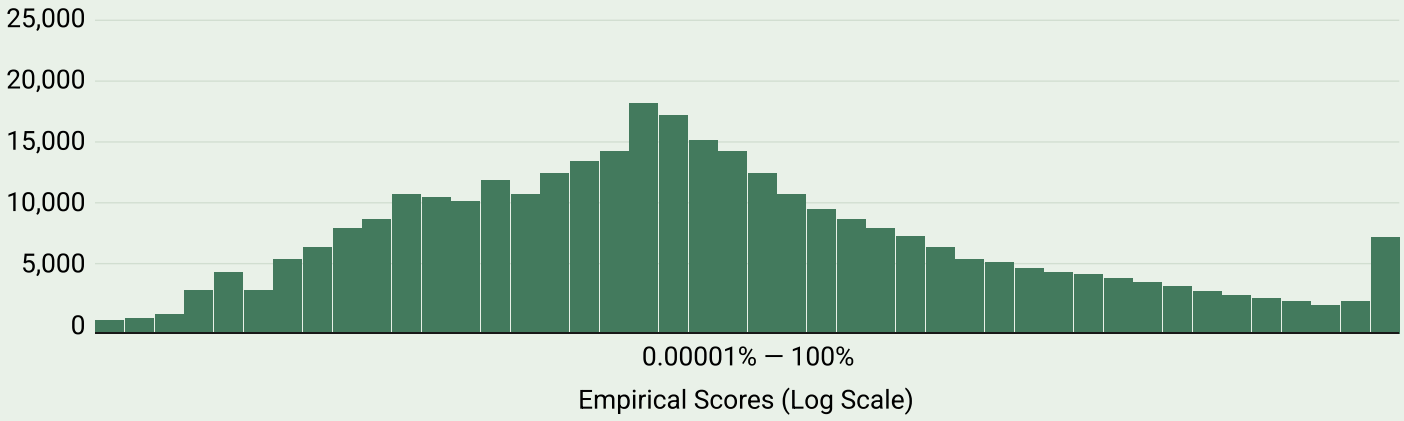
The Empirical Global Model

The Empirical Global Model is a production model that assigns each published CVE a calibrated 30-day probability of observed exploitation activity (based on our telemetry coverage).

The goal is practical: help vulnerability teams focus limited time on the CVEs most likely to face near-term attacker attention, while being clear about what the score does not claim (it is not impact, not asset-specific risk) and what it is: a probabilistic statement that provably describes attacker behavior better than any other available strategy.

To decide whether any model like this is "good," you need a few things to be true at the same time. You need to understand what decision the model supports (so it's used correctly), what evidence it relies on (so inputs are credible), how it turns evidence into a score (so it isn't magic), and how well it performs compared to common baselines like severity-only approaches. Finally, because the threat landscape moves, you need to know how the model is monitored and kept reliable over time, so trust is based on ongoing reality—not a one-time result.

Global Vulns Distribution by Empirical Score



EMPIRICAL

How the Empirical Global Model Supports Better Vulnerability Management

Most vulnerability management teams are trying to answer the same practical question: "Which vulnerabilities should we work on first?" The challenge is that everything can look urgent when you have more findings than capacity.

This model helps by providing a rigorous and science-backed global view of attacker attention and activity. It's a very efficient way to separate "high likelihood in the wild" from "probably not seeing active pressure right now." Used well, it reduces noise and helps you focus scarce remediation effort where it is most likely to matter soon.

Just as importantly, the score is designed to plug into broader risk thinking:

Likelihood (this model): "How likely is this CVE to see exploitation activity soon?"

Exposure and controls (your environment): "Is the vulnerable thing present and reachable, and do we have defenses?"

Impact (your environment): "If it happens here, what does it mean for us?"

This is why the Global score is a good baseline: it answers the "outside world" part consistently, even when local data is incomplete.

Using the Global model "as is" vs adding local context

Not every organization has the same tooling, asset visibility, or budget. The model supports both ends of that spectrum.

If your VM program is still maturing, you can use the Global score "as is" to drive prioritization. This is a big step up from severity-only or vendor-based sorting because it is anchored to observed attacker activity patterns. Many teams start here and immediately get a tighter, more workable remediation queue.

If your VM program is more mature, you should treat the Global score as the first layer and then add local context before acting. Examples of local context include:

Is the vulnerable software actually present in your environment?

Is it internet-facing or reachable from untrusted networks?

How critical is the system (business function, data sensitivity)?

What compensating controls exist (segmentation, EDR, WAF/IPS, patch windows)?

Are there internal detections, scans, or incidents tied to this exposure?

This document does not prescribe how you must do that enrichment (that depends on your tools and processes), but the principle is simple: Global likelihood tells you where pressure exists; local context tells you where you're vulnerable to that pressure.

How the global model connects to your workflow

Most teams follow the general pattern:

1. Adjust the probability (score) for local guardrails (presence/exposure/criticality/controls). For example set probabilities to 0 when you're certain a vulnerability is not present in your system.

2. Filter/Rank by probability, typically with one of these strategies:

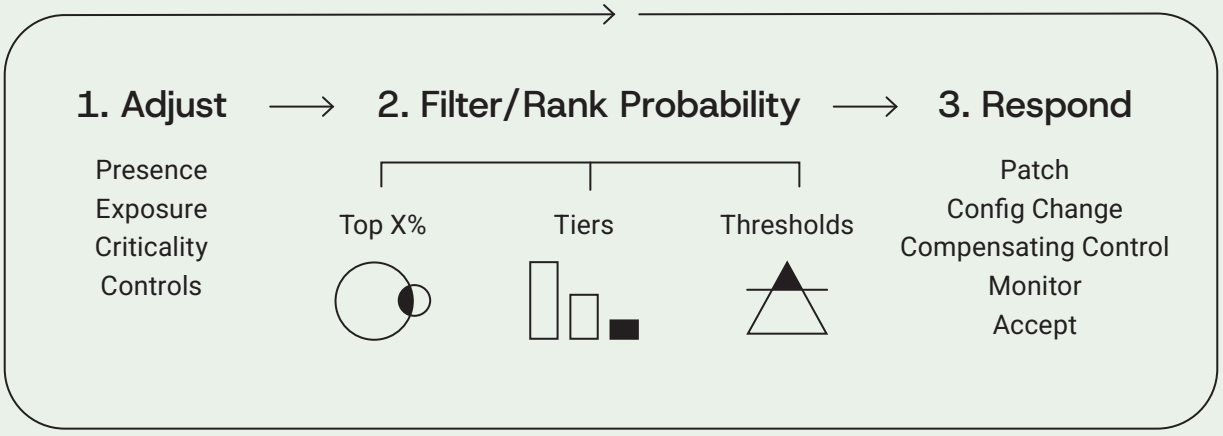
Top X Percentile up to set capacity. Everyone appreciates structure and filling a set capacity may reduce "fire-fighting" mode. Always work the top 10-15 percentile each cycle to fill a fixed capacity.

Tiers map ranges to targets (e.g., high/medium/low with response windows). Best for SLA-style programs.

Thresholds work anything above a cutoff (e.g., ≥0.10). Best when you want a simple rule, but there is no threshold at which nothing below may be exploited in the future.

3. Choose the response (patch, config change, compensating control, monitor, accept)

The Empirical Global Model is designed to be easy to adopt: teams can start with a simple Top X percentile, threshold, or tiering approach and get value immediately. Because the score is grounded in observable external evidence, it also gives security teams a common, defensible basis for prioritization—making it easier to explain tradeoffs, resolve internal disagreements, and withstand "why are we fixing this one?" challenges without relying solely on severity or intuition.



EMPIRICAL

Model Inputs: What We Measure

The Empirical Global Model is built from observable, external evidence about vulnerabilities. We collect signals that indicate attacker attention, weaponization, and real-world environmental activity. Inputs are collected from multiple sources, aggregated and transformed into a consistent feature set, and then refreshed as frequently as sources allow (often hourly, sometimes daily). We currently use 2,823 features in production across the categories below.

Model Features

These categories are for explanation only. They are not separated in the model, nor do they represent separate models.

Chatter: Mentions of CVEs in articles, blogs, and social sources, plus derived trend signals (e.g., spikes vs steady background).

Exploit Code: Indicators that exploit code exists or has become available in the ecosystem.

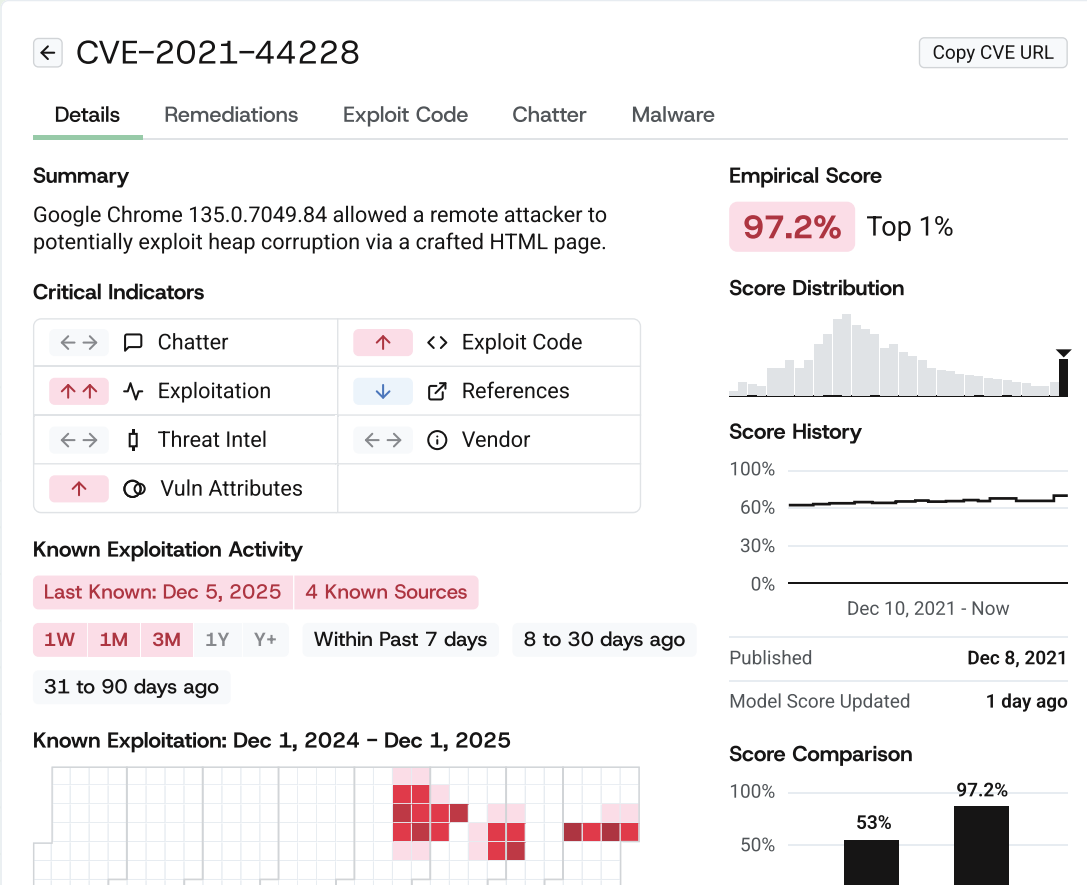
Exploitation Activity: Telemetry-derived indicators of observed exploitation activity, including lagged/auto-regressive summaries of recent and historical activity (built with strict time cutoffs).

References: The amount and types of references associated with a CVE across vulnerability knowledge sources (used as a proxy for structured attention and corroboration).

Threat Intel: Signals from tooling/scanner ecosystems and other industry indicators that reflect operational focus on specific CVEs.

Vendor: Normalized vendor/producer signals (at scale) to capture ecosystem differences across major vendors and product families.

Vulnerability Attributes: "Intrinsic" vulnerability descriptors, including CVSS vector components, CWE class, and derived tags extracted from descriptions to capture prerequisites, outcomes, and weakness characteristics.



Ground Truth (outcomes)

Exploitation Activity is the model's closest link to real-world attacker behavior. We collect exploitation telemetry from a mix of source families (both open and commercial), with the bulk of coverage coming from network-based detections, and additional signals derived from endpoint and malware-analysis sources.

These data do not claim to capture every exploit everywhere; they represent what is observable through the telemetry we have. That's why we describe the model outcome as "observed exploitation activity," and why "no observed activity" should be read as "not observed in our coverage," not "did not happen."

In the model, exploitation telemetry is used in two ways. First, it defines the outcome we're trying to predict: whether exploitation activity will be observed (within the model's coverage) over the prediction horizon. Second, it provides lagged (auto-regressive) signals that summarize recent and historical activity—for example, whether activity was seen very recently, whether it has been persistent, or whether it appears in bursts. To keep the prediction honest, we enforce strict time boundaries: features are computed as-of 00:00 UTC and only use information available before the prediction window, so the model learns from the past without accidentally "peeking" into the future.

Time consistency and eligibility (why inputs are trustworthy)

The "unit of measurement" in the model is a CVE in a PUBLISHED state. We treat CVE Program status (via cve.org) as the source of truth for whether an identifier is eligible for scoring. Daily feature snapshots are constructed as-of 00:00 UTC, so each row represents what was knowable at that point in time. We also have a strict "no overwrite" policy and have built a robust point-in-time architecture. This enables us to reconstruct "what the system knew" at a given time and replay processing when upstream formats change or errors are discovered.

Data quality controls (simple, practical)

We monitor more than just errors in execution, we implement layers of "watchdog" processes that monitor for data "freshness" that the expected volume of data and record is consistent, various completeness, quality and consistency checks. These all exist to reduce the likelihood that "silent errors" creep into the data stream and thus into the modeling.

> *"Global likelihood tells you where pressure exists; local context tells you where you're vulnerable to that pressure."*

How The Models Learn

The Empirical Global Model learns from history in a straightforward way: it looks at what we could observe about each vulnerability on a given day (presently & historically), and learns which patterns and indicators tend to be followed by observed exploitation activity.

It then uses those learned patterns to produce a calibrated 30-day probability based on what we know right now about each published CVE. In plain terms: we train on data where each row is a CVE on a specific date ("what we knew then"), paired with ("did this lead to exploitation activity").

Learning engine: XGBoost (simple but serious)

We use XGBoost, a canonically used gradient-boosted decision tree method. It builds a strong predictor by iteratively adding many small decision trees, each designed to improve the mistakes of the combined model built so far. A decision tree is a set of if/then splits on input features that routes each daily observation to a leaf value; that leaf contributes a small numeric adjustment to the overall score. By adding many such trees, the model learns nuanced combinations of evidence—patterns that are difficult to capture with a single threshold or a single severity score.

For example, the model can learn rules that resemble:

If indicator A is present and indicator B is high, the likelihood increases.

If indicator C is absent, but indicator D shows recent activity, the likelihood still increases.

Any single rule is limited on its own, but many rules combined allow the model to capture the complex, non-linear relationships that appear in real-world security data.

Why we chose this approach

We evaluated several model families including elastic-net logistic regression, random forests, support vector machines, and neural networks. We selected XGBoost because it provided the best combination of predictive performance for this problem (many features, sparse signals, and meaningful interactions between indicators).

From model score to a 30-day probability (calibration)

The raw output of the model is a score that is good for ranking, but we want it to behave like a probability. To do that, we apply polynomial basis expansion method, which learns a simple monotonic mapping from raw scores to calibrated probabilities. Calibration is what allows us to say: "among vulnerabilities with a predicted probability of ~5%, we observe exploitation activity at roughly that rate (within expected statistical variation)."

This calibration step is where we align daily learning signals to the 30-day probability used in production.

Validation approach

We validate in two complementary ways. First we set up a time-split evaluation. We train (execute the XGBoost algorithm) on earlier time periods and then we validate and test how well the model can predict later outcomes that were withheld from during training. This mimics the model's use in production: you want it to perform well in the future, but it can only base its decisions on present information.

Second, we remove a random sample of vulnerabilities (the entire history of a sample of CVEs) from the training data and do specialized testing on those specific vulnerabilities. This allows us to train models that can **generalize** to vulnerabilities that they haven't seen before, which is important since in the real world, new CVEs are published every day.

We use a technique called cross-validation where we train many models on rotating time periods and withholding different sets of CVEs. These procedures help us reduce the likelihood that we are overfitting (memorizing the dataset instead of understanding underlying patterns).

Validation

The Empirical Global Model is designed to be trusted the same way you would trust any operational measurement system: it makes a clear, testable claim, it is evaluated against real outcomes, and it is monitored continuously as conditions change.

In practice, we validate the model in two ways: (1) does it help teams focus effort better than common baselines?, and (2) does the score behave like a probability over time?

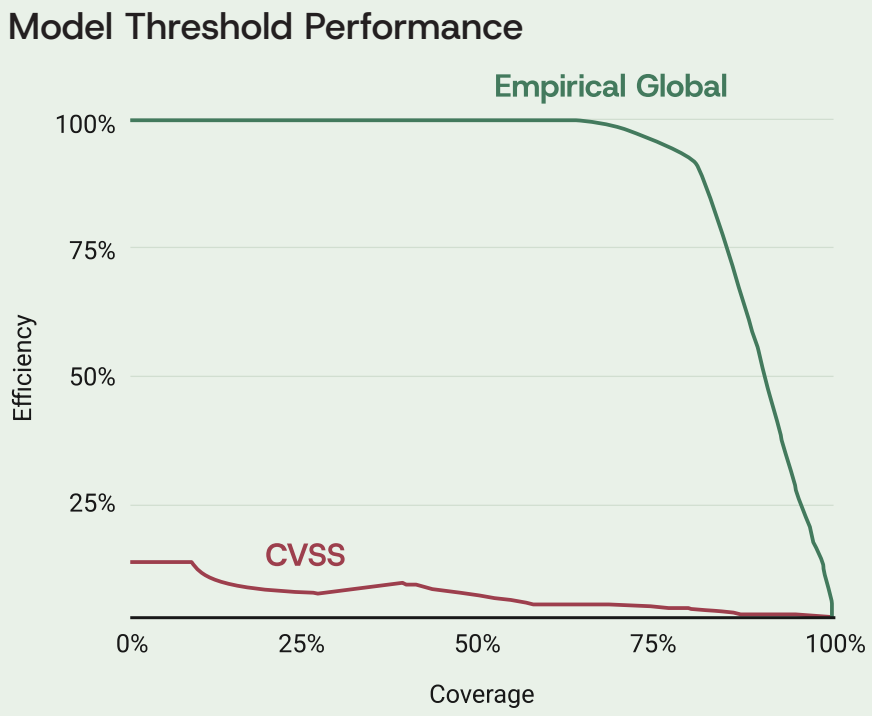
Does it prioritize better than CVSS and vendor scores?

CVSS is useful for describing technical severity, but it was not designed to predict near-term exploitation activity. Our model is specifically trained to forecast observable exploitation activity, so we expect it to be more efficient at prioritization.

Coverage (Recall): of all CVEs that later show exploitation activity, what fraction did we prioritize?

Efficiency (Precision): of the CVEs we prioritized, what fraction later show exploitation activity?

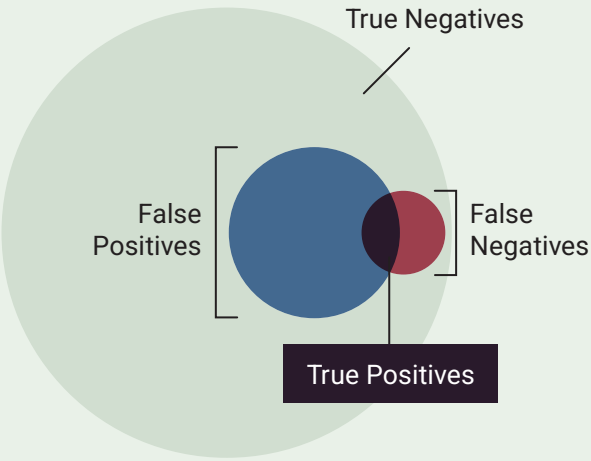
Effort: the proportion of all PUBLISHED CVEs you choose to prioritize (e.g., top 5%, top 10%).



How to interpret it:

At any fixed level of effort (say, “we can address 10% of published CVEs”), a better system either catches more of the exploited items (higher coverage), or wastes less effort on items that don’t show activity (higher efficiency). Compared to CVSS, it clearly improves both efficiency and coverage.

All Published Vulns
(CVEs we could prioritize)



Effort

Measures the relative workload as the proportion of vulnerabilities prioritized out of all the possible CVEs:

True Positives + False Positives / Everything

Coverage

Measures how well our strategy covers the vulnerabilities we prioritized from all vulnerabilities that show exploitation activity (False Positives are prioritized without any exploitation activity observed):

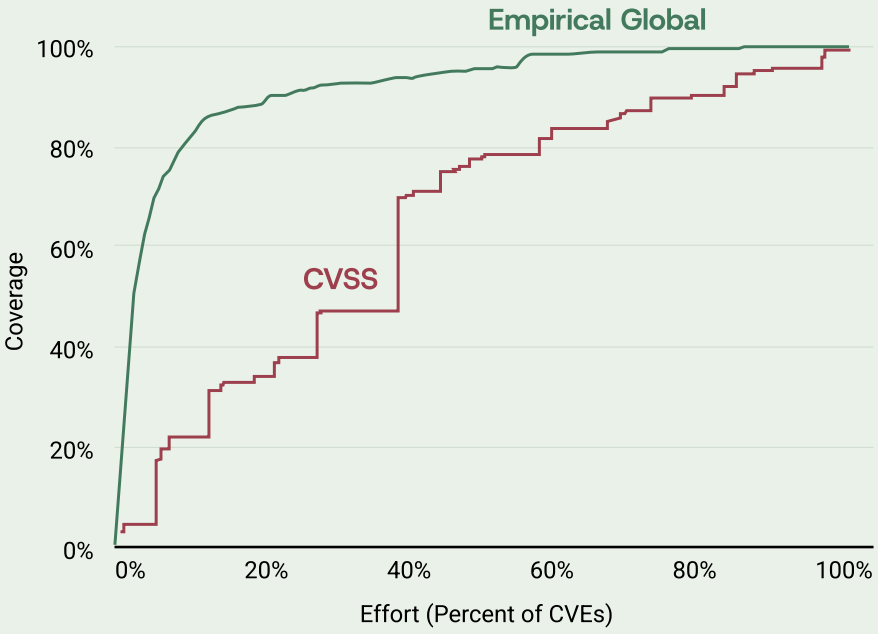
True Positives / True Positives + False Positives

Efficiency

Measures how accurately our strategy focuses on vulnerabilities that have exploitation activity (False Negatives are not prioritized and exploitation activity is later observed):

True Positives / True Positives + False Negatives

Coverage vs Effort (the capacity view)



The Coverage vs Effort curve is the most intuitive chart for practitioners because it answers, "If we can only work on X% of the backlog, how much of the observed exploitation activity do we cover?"

Understanding of this chart will help you understand the tradeoffs inherent in any prioritization strategy.

Probability you can actually interpret (calibration)

Because the output is a calibrated probability, we also validate that the score behaves like a probability. The simplest way to show this is a calibration plot: group CVEs into probability buckets and compare predicted vs observed exploitation rates. A well-calibrated model makes the score usable for thresholds and tiers because "0.10 means about 10% under this definition and coverage."

Ongoing validation and monitoring (why trust holds over time)

Security data changes constantly. A model isn't trustworthy because it performed well once—it's trustworthy because it is re-tested and monitored in production. Our monitoring focuses on four practical areas:

Data health: freshness, volume/missingness, and distribution shifts across key inputs (to catch silent feed issues).

Score health: checks for unusual shifts in score distributions (e.g., everything suddenly becomes "high risk").

Performance health: periodic backtesting on new time windows to confirm the model continues to separate higher-likelihood from lower-likelihood CVEs.

Calibration health: confirming the probability interpretation remains stable (or re-calibrating when necessary).

This is a key difference from static approaches: many common scoring methods (including severity-based sorting) do not have a feedback loop that checks whether the score matches observed reality. Our approach is explicitly built around learning from outcomes and continuously checking that the relationship still holds.

Discussion: why this approach is more trustworthy than “rules”

A rules-based prioritization system (or a pure severity sort) has two predictable problems:

1. It often can’t represent complex interactions (“this is risky only when these two conditions hold”), and

2. It rarely comes with a built-in way to measure whether the rule still matches today’s threat environment.

By contrast, this model is trained directly on observed outcomes, validated against baselines, and monitored so shifts in data quality or adversary behavior are detected and handled. That combination of clear claim, empirical validation, and continuous monitoring is what makes the score operationally trustworthy.

EMPIRICAL

Empirical builds data-driven models for security teams. We're building the world's first local models for cybersecurity, we maintain the world's most advanced global models, and we power existing open-source technology — EPSS, used by over 120 vendors today.

See how your model would differ — book a walkthrough
www.empiricalsecurity.com | info@empiricalsecurity.com