

The Bayescope methodology

Every number this product shows you was produced by arithmetic you can read, from inputs it recorded, with a seed you can re-run. This document says how — precisely enough that someone who does not trust us can check.

Version 0.2.1 · updated 2026-10-07

Contents: Why this document exists · One loop, eight steps · The mathematics · Why deterministic mathematics, and not a model's judgement · Where the numbers come from · The limits of this method · Colophon

Why this document exists

Risk numbers are easy to produce and almost impossible to defend. A scoring tool gives you an 82 out of 100 and cannot tell you what would have made it an 81. Under NIS2 and DORA that is no longer an academic complaint: management signs for the result, and a signature over a number nobody can derive is a liability.

Bayescope does not claim its numbers are right. Priors drawn from public incidents are estimates, and estimates are wrong in ways that look exactly like being right. What it claims is narrower and more useful: **every number is reproducible, traceable, and falsifiable**. Reproducible, because the run stores its seed and the complete state of its inputs, and the same inputs give a bit-identical answer. Traceable, because the prior, each piece of evidence, each likelihood ratio and each posterior are persisted in an append-only trail that recomputes. Falsifiable, because the content that produced the number is published to the people who bought it — this document, the scenario library, the sources on every prior — so it can be argued with.

That is the whole product argument. The rest of this document is the mechanism behind it.

One loop, eight steps

There is one methodology engine. Assess, Advise, Operate, Respond and Emulate are five ways into it, not five products. The loop runs continuously, and each complete pass is snapshotted immutably so this quarter can be compared with the last.

01 • Observe

Bayescope builds a model of your environment as a directed graph: assets, identities, the reachability and trust relations between them, and the controls that apply to each. Nodes arrive from read-only connectors to systems you already run — identity, endpoint, SIEM, SOAR, vulnerability management, network — from CSV import, or from a consultant typing what a client told them.

Every node and edge carries its provenance, a confidence weight and a last-seen time, because the graph is a belief about the environment, not a statement of fact. A machine-reported asset outranks a human-claimed one. An entity merely named inside an alert — a username in a log line, an internal address in an event — is recorded as **proposed** and stays outside the graph until a person confirms it. Analysis therefore never runs on entities nobody has ever checked, and confirming a device changes the model deliberately rather than as a side effect of a sync.

Interfaces are treated with the same care. Every network interface of a machine is recorded so an address in a log resolves to the right host, but only a real one may become an attachment point. A container bridge admitted as an attachment point would put a private container network into the model as a segment the estate can be reached across, and manufacturing reachability that does not exist is worse than missing a pivot that does.

02 • Detect anomalies

Bayescope does not run its own detection analytics on your traffic, and does not place an agent on your endpoints. It reads what your detection stack already found. Alerts arrive through a single guarded intake — one place in the code where an external batch becomes internal records — and every guarantee is enforced there once, for every source, rather than in each connector.

An alert becomes an anomaly and an observation. Where a mapping table recognises the alert text, the observation names the exact scenario indicator it evidences; where it does not, the tracker falls back to a conservative textual match. Either way the posterior only ever moves for a reason someone can inspect.

Two rules keep the arithmetic honest. A pull connector returns every still-open alert on every sync, so a re-sighting of an alert already seen is recorded as an update and **never re-applies its likelihood ratio** — the same evidence counted twice is not more evidence. That holds across systems too: a SOAR incident that names the SIEM offense it came from is the same incident, joins the same case, and moves a posterior once between them, whichever of the two arrives first. A hunt is treated the same way: twenty hits of one query are one test of an indicator, not twenty. And where the product's own sources report what they observe — a new unrecognised device on a network, say — an indicator may be moved by automation at most once per window. Twenty-four unreviewed devices arriving on a busy network is one fact about that network, not twenty-four independent tests of it; recording all twenty-four but applying the likelihood ratio once is what keeps a posterior from ratcheting to a certainty it can never come back from.

These rules are enforced where the update happens, not in each source. The one function that applies a likelihood ratio refuses to be called without stating how repeats of its observation count, so a connector written next year cannot forget.

03 · Find the binding constraint

A system's behaviour is shaped by its chokepoints, not by the sum of its parts. Bayescope computes the attacker-side chokepoints exactly: the **minimal cut set** of intermediate nodes whose removal disconnects every internet-facing entry point from every crown jewel.

Each chokepoint is then ranked by the fraction of sampled attack paths that pass through it, and the evidence for the ranking — how many paths, out of how many sampled, and up to three of them written out end to end — travels with the number, so the answer to "why does this bind?" is on the same screen as the claim that it does.

The defender side is the other half of the same idea, and is just as real a constraint: staff hours per month, the patch window, the budget, the systems that cannot be touched, the change freeze. These are declared, typed, and are what the investment optimiser is actually solving against.

04 · Adapt analogues

Attacks are not novel as often as vendors need them to be. The scenario library encodes public incidents as **structural templates**: what had to be true, in what order, for the thing to happen. Each template names the incidents behind it, the actor class, the public reference, and the structural lesson drawn from it.

An analogue is not a prediction. It is a shape that has been observed to work, held against your environment to ask whether the same shape fits. The library is currently twenty templates spanning ransomware, edge exploitation, business email compromise, supply chain, cloud credential theft, web application attack, insider action, physical compromise, and a set that fits none of those neatly — helpdesk social engineering, denial-of-service extortion, domain hijack, an internal worm.

05 · Generate scenarios

A template becomes a scenario instance by being evaluated against one client's graph.

Preconditions are written in a small declarative expression language with a real parser — a tokeniser and a recursive-descent grammar, never a call to a general-purpose evaluator — so an expression the grammar does not recognise is refused when the content loads, not executed at runtime.

The logic is three-valued, and this is where most tools quietly cheat. A condition can be true, false, or **unknown**, and unknown is never silently read as false. A required precondition that cannot be verified does not block the scenario: it instantiates with the caveat recorded and the prior dampened, because absence of telemetry is not evidence of absence. A context multiplier whose condition is unknown falls back to the template's declared default, never to the branch that would raise the risk. Bayescope does not amplify a number on data it does not have.

Some multipliers ask about the world rather than the estate — is this sector the target of a current campaign of this shape? Those are filled from threat- intelligence feeds you choose to connect, as facts with an expiry: the multiplier applies while the campaign is being reported and returns to its default when nobody re-reports it. A campaign's beginning is announced far more reliably than its end, and a membership that never lapsed could only ever raise a number.

Stage roles then bind to actual assets through a canonical vocabulary of asset classes. A source that reports a device as an "endpoint" has not told us whether it is a workstation or a server, so that word is not treated as a synonym for either. Where a role binds to nothing, the screen says so rather than showing a confident number computed against an empty set.

06 · Rank by Monte Carlo

Each instantiated scenario is compiled into a staged chain and simulated. See the mathematics below for exactly what is computed. The output is a probability that the campaign reaches its objective, an interval around it, a ranked sensitivity showing which stage the answer actually turns on, a dwell-time distribution, and a loss distribution — and the SHA-256 hash of the complete input state that produced all of them.

07 · Act

A finding becomes a decision by asking what would change if you did something. The counterfactual re-runs the same scenario with a candidate control made effective, **under the same seed as the baseline**, so the difference between the two runs is the control's effect and not the random number generator's opinion. A delta whose confidence intervals overlap the baseline's is reported as no measurable effect at this sample size, never as a win.

Candidate initiatives are then scored by expected loss removed per euro across the whole scenario portfolio, and selected under your declared budget, staff hours, dependencies and untouchable systems. An initiative with no measurable risk reduction is never bought, even when it fits the budget comfortably. Recommending that is the failure mode this product exists to prevent.

Where response is in scope, an action's expected effect is modelled the same way before anyone approves it, so the approval screen shows the pre-flight delta beside the button.

08 · Update

New evidence moves the posterior; the loop closes. Observations are matched against each scenario's declared indicators and the likelihood ratios are applied in odds form, oldest evidence first. Falsifying indicators are applied identically: evidence that lowers a probability matters exactly as

much as evidence that raises it, and a library template with no way to be argued down is a template that only ever ratchets up.

Silence is handled carefully. Where an expected high-confidence indicator has not appeared within its window, that silence decays the posterior slightly — but only for stages you actually monitor. Decaying a stage you have no telemetry for would convert blindness into comfort, which is the precise opposite of the point.

A complete pass of the loop is written as an immutable cycle snapshot. Each scenario in it carries both numbers: the simulated prior, with its sampling interval, and — where evidence has moved it — the evidence-updated posterior, which is the headline the snapshot ranks, prices and reports. Reports are assembled from a snapshot, never from a live query, and cannot be exported until a named human has signed the draft. What you hand your auditor is a specific artefact, tied to a specific state of the world, with a specific person's name on it.

The mathematics

Three pieces of unglamorous, decades-old mathematics. None of it is novel, and that is the point: an auditor can check it against a textbook rather than against our description of it.

Bayesian updating, in odds form

Belief is updated by converting to odds, multiplying by the likelihood ratio, and converting back:

$$\text{posterior_odds} = \text{prior_odds} \times \text{LR}$$

Odds form is used throughout rather than the probability form of Bayes' rule for three reasons. It composes — several independent pieces of evidence are a product of their ratios. It is order-independent for independent evidence, so the answer does not depend on the sequence the connectors happened to sync in. And it cannot leave the valid range the way naive multiplication of probabilities does.

Every likelihood ratio comes from the scenario template, authored with its source. An indicator observed is not a fact about the world; it is a statement of how much more likely that observation is when the scenario is running than when it is not, which is exactly what a likelihood ratio means.

Matching an observation to an indicator is deterministic. Either the observation explicitly names the indicator — the precise path, used by connectors, hunts and detection-rule exports — or at least 60% of the indicator's distinctive terms appear in the observation's text. The threshold is deliberately conservative in the same direction throughout: a missed match costs one probability update, while a false match corrupts the audit trail.

Probabilities are held strictly inside zero and one. A posterior that reached either bound could never be moved by any future evidence, and nothing in security is ever epistemically certain.

The whole chain — initial prior, every step, current posterior — is persisted append-only, at the object layer and again as database triggers, and it recomputes: replaying the arithmetic must reproduce every recorded step to within one part in a billion. An altered number does not reconcile with the step before it. Note what this is and is not: it makes tampering with a stored derivation detectable by recomputation, and it prevents modification and deletion through the application and the database. The derivation rows themselves are not a cryptographic hash chain. The audit trail around them is: since September 2026 every audit entry — each observation added, each sign-off,

each configuration change — is hashed onto its predecessor, one chain per client, and an auditor's package carries a verifier that runs on a stock Python with nothing of ours installed. Entries written before the chain existed are pinned by a digest inside it rather than backfilled, because a hash computed today over a row read today attests to nothing about the day it was written.

Monte Carlo, and what the interval means

A scenario is a sequential chain of stages: reaching stage k requires succeeding at every stage before it. Each trial samples each stage's base success probability from the distribution the template declares — beta, lognormal, triangular, normal, uniform, or a point value — and the effective probability for that stage in that trial is

$$p = \text{base} \times \text{context multipliers} \times (1 - \text{control reduction})$$

where controls combine as independent failures, $1 - \prod(1 - e)$. Two controls that are each 50% effective leave 25% through, not zero. Defence in depth helps and never reaches certainty; a model that let a stack of controls sum to 100% would produce exactly the false comfort this method is built to avoid.

Each template also states one number outright, with its source: the probability that a campaign of its shape, once attempted against a typical organisation, reaches its objective. The stages must **decompose** it — the product of their mean success probabilities has to equal that headline within a stated tolerance, or the template does not load and an edit does not save. The headline is the claim a reviewer argues with; the stage split only says where the difficulty lies.

Ten thousand trials run by default. The proportion that reach the objective is the reported probability, and around it Bayescope reports a **Wilson score interval at $z = 1.96$, which is a 95% interval**. Wilson rather than the normal approximation because the normal approximation produces impossible bounds near zero and one — below zero, above one — and security probabilities live near zero. Anything in this product describing that interval as 90% is a defect; the level is 95%.

The interval is a sampling interval. It describes how precisely ten thousand trials pinned down the probability implied by the inputs. It says nothing about whether those inputs are right, and it is never lent to a number it does not belong to: a Bayesian posterior derived from evidence has no sampling interval, and is shown without one rather than borrowing the simulation's.

Sensitivity is computed analytically rather than by re-running. For a sequential chain the probability of the objective is the product of the stage probabilities, so the derivative with respect to one stage is P / p_k — exact, free, and it identifies the stage the whole answer turns on. Dwell time and loss are reported over successful runs only, as percentiles, and expected loss is the probability of the objective multiplied by the mean loss given success.

That figure is per attempted campaign. A yearly figure needs to know how often such a campaign is attempted, so each template declares an attempt frequency as a range a reviewer can disagree with directly. Annual loss is then simulated year by year: attempts drawn from that frequency, successes from the probability, a magnitude drawn for each success. That is what makes the tail of the curve mean a two-campaign year rather than one expensive one. A template that declares no frequency gets no annual figure, and the screen says why rather than assuming one attempt a year.

The complete input state of a run — engine version, template and its version, run count, seed, every stage's distribution and its source, every multiplier and its source, every control reduction — is serialised canonically and hashed with SHA-256. That hash is stored with the result. Re-running from

the stored inputs is what "reproducible" means here, and it is checked by tests that must never go red.

Binding constraints as minimal cut sets

The chokepoint question has an exact answer, so Bayescope computes one rather than scoring it. The graph is transformed by node splitting: each node becomes an in-node and an out-node joined by an edge of capacity one, and every original edge is given infinite capacity. Entry points and crown jewels get infinite capacity too, because they are the endpoints of the question rather than candidate chokepoints. A minimum cut on the transformed graph can therefore only pass through intermediate nodes, and it is a minimal set of nodes whose removal disconnects every entry point from every crown jewel.

Two honesty rules follow from properties of the mathematics rather than from policy. **Minimum cuts are not unique** — a graph may have several cuts of the same size — so the product reports a minimal cut and ranks chokepoints by the fraction of sampled paths through them; the tests assert the guarantee, not one particular cut. And path enumeration is bounded: up to two thousand paths, none longer than ten hops. When the sample is truncated the truncation is reported with the result. A silently capped sample is a wrong number that looks like a right one.

Where a crown jewel is itself an entry point, or sits one edge from one, there is no intermediate node to cut and no finite cut exists. That jewel is returned as its own chokepoint, which is the true answer and a useful one.

Counterfactuals, and risk removed per euro

The counterfactual is the same simulation with a control set made effective, sharing the baseline's seed. Its significance test is deliberately strict — the intervals must not overlap — because an insignificant delta presented as a result is how security budgets get spent on nothing.

Across the portfolio, an initiative's score is the expected loss it removes divided by what it costs. Selection then solves a constrained knapsack: budget, staff hours, dependency closure, untouchable systems. The ordering invariant is that no measurable effect always ranks below a measurable reduction, whatever the price.

Why deterministic mathematics, and not a model's judgement

A language model is very good at reading a policy document, and structurally unable to produce a number anyone can audit. Ask it the same question twice and it may answer differently. Ask why, and you get a plausible reconstruction rather than a derivation. The number carries no interval, no provenance and no seed. Worst of all it cannot be checked by someone who distrusts the vendor, which is precisely the person a regulated organisation has to satisfy.

So the division of labour is absolute, and it is enforced in code rather than asked for in a prompt. A prompt is a request; a validator is a guarantee.

Deterministic Python	Language model
Priors, likelihood ratios, control efficacy	Reading documents and extracting claims

Deterministic Python	Language model
Monte Carlo, intervals, sensitivity	Drafting report narratives
Bayesian updating, the evidence chain	Explaining a result already computed
Minimal cut sets, path ranking	Recommending a triage disposition for a human
Counterfactuals, optimiser scoring, loss	Turning prose into a draft scenario structure

What the language model may never do

Model output is validated before it reaches anything. Fields the engine owns are rejected at any depth of nesting, so a probability cannot arrive smuggled inside a nested object. Prose asserting a quantity attached to a risk word is rejected in both word orders — "the probability is 34%" and "a one in five chance" are the same violation. Every factual sentence in generated prose must carry an evidence reference or be explicitly flagged as an assumption, and a model that cannot meet that gets one corrective attempt carrying the validator's exact complaints, then fails visibly and leaves the work with the human.

The structural defences matter more than the textual ones. When a document is parsed into control claims, those claims are forced to attested in code after the model returns — so a document engineered to convince a model to say "verified" still cannot produce a verified control. Where the model drafts a scenario, it drafts structure only; every prior and likelihood ratio is set by the analyst, with a source, before the version is saved.

And when no model is available, AI features refuse with a stated reason. They never silently degrade, and they never fabricate. Every AI task declares the capability tier it needs, so a small local model refuses work it cannot do rather than producing confident rubbish.

Verified, attested, unknown

A control is in exactly one of three states, and the distinction is enforced at the object layer and by a database constraint, not by convention:

- **Verified** — machine evidence exists, and the reference to it is stored. A verified control cannot be written without one.
- **Attested** — a human says so. An uploaded policy document is a claim, not proof; storing it does not change any control's state, and recording what it shows is a separate, audited act whose source is forced to "document".
- **Unknown** — nobody has said, and nothing has reported. Shown as unknown, never rounded down to compliant or up to a gap.

There is no path from attested to verified, and the codebase pins that structurally: the one place that can construct a verification is enumerated by a test, so a playbook, a ticket, a remediation or an auditor's review can never write it. An auditor's finding of "compliant" does not upgrade a control either. Only machine evidence does.

Assumptions are registered, not buried

Any prior, multiplier or default that affects an output is recorded in an assumption register with its source and its rationale, and comes up for review on a schedule. When a scenario instantiates with an unverifiable precondition, the dampening that was applied is itself registered as an assumption with the caveat attached. There are no magic numbers in business logic; if a constant affects a result, it is either content with a stated source or a structural constant documented where it lives.

Where the numbers come from

Content, not code

Scenario templates, the control taxonomy, framework mappings, regulation packs, the connector catalogue, the alert map, the asset-class vocabulary, response playbooks: all of it is versioned YAML loaded at runtime, not compiled into the product. This is deliberate and it is the moat. A methodology that requires a software release to correct a prior is a methodology nobody will correct.

Every file is validated twice at load — once against a JSON Schema, which is the authoring contract and gives file-and-field error messages, and once as typed objects the engine consumes. A file that passes the first and fails the second means the schema and the model have drifted, and that is reported as loudly as bad content. Cross-references are checked too: a framework mapping pointing at a control that does not exist fails the load rather than silently dropping a requirement, and a scenario indicator named by an event mapping must exist in the library, because a typo would silently stop a number moving forever.

The packs are honest about their scope. Framework packs — CIS Controls v8, NIST CSF 2.0, ISO/IEC 27001:2022 Annex A — are curated subsets and each declares so in a coverage note. The regulation packs are a base NIS2 pack, a DORA pack, and twelve country-gated national packs: eleven NIS2 transpositions, plus Switzerland's ISG, which is not a NIS2 transposition and says so in its name. Where a national law is still a draft, the pack says "pending" in its own name, because a bill must never read as law. Every pack carries a legal disclaimer, and Bayescope does not render legal opinions.

A review binds to a hash

The library ships flagged as needing expert review, and a review is not a permanent blessing. Signing off a template records the SHA-256 of the exact content reviewed. Any later edit changes that hash, and the status reverts automatically to needs re-review — a review of version one never silently blesses version three. The trail of who reviewed what, and when it lapsed, is append-only and is never rewritten.

Clients may author their own scenarios. An authored scenario analyses immediately, so it is useful the day it is written, and is **excluded from signed reports** until an expert has signed its content hash — and the draft states how many were excluded and names them. Editing one reverts its review, for the same reason as above.

Updates arrive signed, and offline

Content updates ship as a signed bundle rather than a fetch:

- 1 **Ed25519 signature** over the canonical form of the payload, verified against the same vendor key that signs the licence. One trust root; the private half never ships.

- 2 **Per-file SHA-256** inside the signed payload, re-checked after decoding, and a hard path allowlist — a bundle may only write inside known content subtrees, and an attempted path traversal refuses the whole bundle.
- 3 **Staged transactional validation** — files are written into a copy of the content tree and the full loud-failing load runs there. Only a bundle that validates end-to-end is moved into place. A bad bundle changes nothing.

A bundle can be applied from a file on a machine with no internet connection at all. The optional pull channel is off unless an address is configured, which keeps the zero-egress default intact. And when a bundle is applied, the running process keeps its in-memory content until it restarts — so the tooling says a restart is required rather than pretending a hot reload happened.

What is still an estimate

The priors and likelihood ratios in the scenario library are conservative expert estimates, sourced against public incident reporting, and every template says so in its own header. They have not been calibrated against a body of engagement outcomes, because that body does not exist yet. Tests prove that the engine is reproducible; no test can prove a prior is correct, and a wrong probability looks exactly like a right one.

This is stated here because it is the honest position and because the mechanism above is designed around it: the priors are content, they carry their sources, they are reviewable, a review binds to what it reviewed, and a correction ships signed without a deploy. Argue with the numbers. That is what they are published for.

The limits of this method

- **A model of an estate is not the estate.** Everything here reasons over the graph Bayescope was given. An asset nobody has ever reported cannot appear in a path, and the product's answer to "how complete is this?" is the provenance and freshness on every node, not a claim of completeness.
- **Minimum cuts are not unique.** A different cut of the same size may be equally correct. Ranking by path fraction is what makes the answer stable and explicable.
- **Path enumeration is bounded**, and truncation is reported. On a large estate the ranking is computed over a sample.
- **Loss magnitudes and attempt frequencies are template content.** They are declared distributions with stated reasoning, not derived from your accounts, and an annual figure is linear in the frequency. Until a template's content has been expert-reviewed, a signed report carries none of its euro figures, and says how many it withheld.
- **Silence is only evidence where you were looking.** Decay applies to monitored stages only, which means an unmonitored stage keeps a stale posterior rather than drifting toward a comforting one.
- **Stage count is not a judgement.** Sequential chains multiply, so a three-stage template once read as structurally less likely than a two-stage one for no stated reason. Every template now asserts its headline number directly and its stages must decompose it, so what orders the ranking is the expert's sourced estimate — which is exactly the thing still awaiting review.

- **Nothing here touches your systems.** Connectors are read-only, adversary simulation runs against the model rather than the network, and the code that performs it is forbidden by an automated structural check from importing anything that could reach a socket, a process, a connector or a model.

Colophon

Bayescope is a product of Anhuret s.r.o. The engine described here is deterministic Python; the mathematics is standard and deliberately old. Where this document and the code disagree, the code is right and this document is a defect — the source of record for every quantity named here is the engine itself.