THREATCRUSH

```
01101100 11010110 10110011
CVE-2026-1042 KEV EPSS:0.94
scan tcp/* 0.0.0.0 iface=eth0
payload= "A'OR 1=1 --" blocked=true
honeypot trap fingerprint=0x9af1
tar-pit drain=4.2KB/s peers=37
```

// PRACTICAL GUIDE

# From Vulnerability Management to Continuous Threat Exposure Manageme

A 90-day operator playbook for evolving from scan-and-ticket VM into a real CTEM loop — without buying nine more tools.

```
module: ctem.discover.asm
priority = exploit * reach * blast
validate exploit -> ok
mobilize alert=slack,sms,webhook
cycle=closed mttm=06:14
```

- The 5 stages of CTEM in operator language
- Why CVSS-weighted backlogs lie to you
- A 90-day implementation playbook
- Metrics that reward outcomes, not activity

threatcrush.com · hello@threatcrush.com · 2026 edition

**A Practical Guide for Operators Who Are Tired of Drowning in Findings**

*A ThreatCrush whitepaper — published 2026*

## TL;DR

Vulnerability management as it has been practiced for the last fifteen years is broken. Scanners produce thousands of findings, prioritization is a wishlist, and remediation queues outlive the engineers who started them. Continuous Threat Exposure Management (CTEM) is the operational answer: a closed loop of **scoping, discovery, prioritization, validation, and mobilization** that runs continuously, not quarterly. This guide explains how to evolve from VM to CTEM without burning down what already works, and how ThreatCrush — a single agent plus a marketplace of security modules — gives small and mid-sized teams the pieces they need to run a real CTEM program without buying nine tools.

## 1. The problem with vulnerability management

The classical VM playbook — periodic scan, ticket-everything, chase a patch SLA — was designed for a world where:

- Attack surface was mostly internal and inventoried.
- Exploits typically lagged disclosure by weeks or months.
- The bottleneck was scanning, not deciding what to do with the output.

None of that is true anymore. Modern attack surfaces include cloud workloads, third-party SaaS, leaked credentials in public repos, abandoned subdomains, exposed admin panels, and supply-chain dependencies updated by people you have never met. The mean time from CVE publication to in-the-wild exploitation is now measured in **days**, sometimes hours. And the bottleneck is no longer scanning — it is the unbearable signal-to-noise ratio of the scanners themselves.

A typical mid-sized environment will produce tens of thousands of "findings" per quarter. The vast majority of them:

- Are not exploitable in context.
- Are not on assets that matter.
- Are duplicates of findings already triaged.
- Are CVSS-prioritized in a way that bears little relationship to actual business risk.

The team triages a fraction, patches a fraction of that, and never closes the backlog. The CISO reports a "patch percentage" that goes up and to the right while the actual exposure window stays open.

**This is the gap CTEM is meant to close.**

---

## 2. What CTEM actually is

Continuous Threat Exposure Management was named and structured by Gartner, but the underlying idea is older: stop treating security exposures as a list of tickets, and start treating them as a continuously updated picture of where an attacker could plausibly get in **today**.

CTEM is not a product category. It is a five-stage operational loop that any team can run, given the right tooling:

1. **Scoping** — agree on what you are protecting and which exposures matter.
2. **Discovery** — continuously enumerate assets, services, identities, and weaknesses across that scope.
3. **Prioritization** — rank findings by exploitability, attacker reachability, and business impact, not raw CVSS.
4. **Validation** — confirm that the exposures you think exist actually do, and that the controls you think are working actually are.
5. **Mobilization** — get fixes shipped: tickets, owners, runbooks, automated mitigations, and follow-up.

The loop runs forever. There is no "done." Each cycle should improve the inputs to the next: a validated finding makes prioritization smarter; a remediated class of bug makes discovery more focused.

---

## 3. The five stages, in operator language

### 3.1 Scoping

Scoping fails most often because security teams scope by tool ("everything the EDR sees") rather than by **business outcome** ("everything that, if compromised, costs us money or trust"). Tool-defined scope has the property that a vendor-renewed scanner suddenly halves your scope without anyone deciding that.

**A useful scoping exercise:**

- List the top five things an attacker could do to your business that would actually hurt — exfiltrate customer data, ransomware the production database, take over the CEO's email, drain the wallet, ship malicious code to customers.
- Trace each one back to the systems and identities that would have to be compromised first.
- That trace **is** your scope. Everything else is secondary.

If your VM tool has 40,000 findings on assets that don't appear in the trace above, those findings are noise until proven otherwise.

### 3.2 Discovery

Discovery in CTEM is broader than vulnerability scanning. It includes:

- Active scanning of your declared attack surface (web apps, APIs, network services).
- Passive monitoring of inbound traffic — every port, every protocol — for things you forgot you exposed.
- External attack surface management (ASM): subdomains, S3 buckets, exposed dashboards, leaked credentials in public dumps.
- Code-side discovery: secrets in repos, vulnerable dependencies, dangerous configurations.
- Identity discovery: dormant accounts, over-privileged service principals, MFA gaps.

The trap here is buying five different "discovery" tools and never reconciling their outputs. CTEM works only if discovery feeds a **single normalized inventory of exposures**, where the same finding from three sources is one entry, not three.

### 3.3 Prioritization

CVSS is a starting point, not an answer. A CVSS 9.8 on a server with no inbound network reachability is less urgent than a CVSS 6.5 on a customer-facing API behind no WAF. Prioritization in CTEM uses, in roughly this order:

1. **Exploitability** — is there a working exploit in the wild? KEV catalog, EPSS score, exploit-DB hits.
2. **Reachability** — can the exposure be reached from where attackers actually are (the public internet, a partner network, a phished employee laptop)?
3. **Blast radius** — if exploited, what does the attacker reach next? Lateral movement potential, credential access, data exposure.
4. **Business impact** — does the affected asset map to one of the top-five outcomes from scoping?
5. **Compensating controls** — is there already a mitigation in place that meaningfully reduces risk?

The output of prioritization is not a CVSS score; it is a small, ordered list of exposures the team should actually work on this week.

### 3.4 Validation

Validation is where most VM programs quietly die. Without validation, the team has no idea whether a finding is real, whether a fix worked, or whether a control is providing the protection it claims to. CTEM treats validation as a first-class stage:

- **Exploit validation** — for top-priority findings, attempt the exploit in a controlled way. Does it actually work in your environment, or is the prerequisite path missing?
- **Control validation** — is the WAF blocking what you think it's blocking? Is the EDR catching the things it claims to catch? Run the test, don't trust the dashboard.
- **Fix validation** — when a fix is shipped, re-run the exact check that surfaced the original finding. Don't close tickets on faith.

Automated, continuous validation is what separates CTEM from "VM with extra steps."

### 3.5 Mobilization

Mobilization is the unsexy stage that makes the rest of CTEM real. It is the work of:

- Routing the right finding to the right owner with enough context to act on it.
- Providing concrete remediation guidance, not "upgrade to the latest version."
- Tracking SLAs that are tied to risk, not to severity letters.
- Triggering automated responses where the risk is high and the remediation is mechanical (rotate the credential, kill the session, block the IP, add the header).
- Closing the loop by re-running validation after the fix.

A CTEM program that nails stages 1–4 and skips mobilization is just a more expensive VM program.

---

## 4. Why traditional tools struggle to deliver CTEM

Most tools in the market today were built for one stage of the loop and bolted into a "platform" by acquisition. The result is:

- A scanner that produces findings but cannot validate them.
- An ASM product that finds exposures but doesn't talk to ticketing.
- An EDR that has telemetry but no view of external exposure.
- A SOAR that can automate responses but has no view of underlying weakness.
- A "risk score" dashboard that aggregates everything and explains nothing.

The integration tax is enormous. Most teams under 200 engineers cannot afford to operate this stack, let alone run a real CTEM loop on top of it.

---

# 5. The ThreatCrush approach

ThreatCrush is built around two ideas:

1. **One agent per server, doing the boring parts well.** A single Linux daemon that handles inbound monitoring on every port, code scanning, automated pentesting against your URLs and APIs, and active defense (tar pits, honeypots, deception). Operators get CLI, TUI, desktop, and mobile clients that all talk to the same agent over an end-to-end-encrypted channel.
1. **A marketplace for everything else.** Discovery sources, prioritization signals, validation checks, and mobilization integrations are modules. Some are first-party. Many are community-published. You install only what your environment needs, and you can publish your own.

This shape is deliberate. CTEM rewards organizations that can compose specific capabilities for their specific scope — not organizations that bought the biggest platform.

## 5.1 How ThreatCrush maps to the CTEM stages

| CTEM Stage | ThreatCrush Capability |
|---|---|
| Scoping | Asset inventory from the agent + module-defined scope tags; you declare which servers and properties matter. |
| Discovery | Built-in network monitor (every port, every protocol), code scanner, pentest engine, plus marketplace modules for ASM, identity, and supply-chain discovery. |
| Prioritization | Findings are normalized and tagged with module-driven severity hints; KEV/EPSS modules layer in exploitability data. |
| Validation | Pentest engine re-runs checks on demand; active-defense modules confirm controls by attempting to defeat them. |
| Mobilization | Real-time alerts (email, SMS, Slack, Discord, webhook), automated active-defense responses, and an API surface for SOAR/ticketing integrations. |

## 5.2 What is different about it

- **One agent, not nine.** The daemon is the unit of deployment. Modules extend it without adding sidecars.
- **Marketplace economics.** Authors get paid for modules, which means the long tail of niche detectors, parsers, and integrations actually exists.

- **Active defense is first-class.** Detecting an attacker is the floor, not the ceiling. Tar pits, honeypots, deception, and automated abuse reports turn detection into cost imposition.
- **Operator-first surfaces.** CLI for muscle-memory work, TUI for live ops, desktop for triage, mobile for "is the building on fire" alerts.
- **Lifetime licensing on the core, usage-based pricing on AI modules.** No subscription treadmill on the parts that don't need one. Pay for AI inference only when you use it.

---

## 6. A 90-day implementation playbook

This is the path most teams should follow to evolve from VM to CTEM. Calendar weeks, not sprints.

### Weeks 1–2: Scope honestly

- Run the top-five-outcomes exercise from Section 3.1 with security, engineering, and at least one business stakeholder in the room.
- Produce a one-page document: critical outcomes, the systems and identities that protect them, and the "everything else" bucket.
- Stop pretending the "everything else" bucket has the same priority as the scoped systems.

### Weeks 3–4: Consolidate discovery

- Inventory every discovery tool currently producing findings. Note overlap.
- Pick a normalization layer — ThreatCrush, a SIEM, or a homegrown table — that will hold one row per exposure regardless of source.
- Wire your top two or three discovery sources into it. Resist the urge to wire all of them on day one.

### Weeks 5–6: Build a real prioritization function

- Document the prioritization formula your team will actually use. Inputs: exploitability, reachability, blast radius, business impact, compensating controls.
- Apply it to the existing backlog. Most of the backlog will drop in priority. Some unexpected items will rise. That is the point.
- Publish the new top-20 list. Get sign-off from engineering on who owns each.

### Weeks 7–8: Add validation

- For every item on the top-20, define a validation check: a script, a pentest module, a control test. Schedule it weekly.
- Wire validation results back into the same normalized exposure store.

- Begin closing tickets only when validation confirms the fix.

### Weeks 9–12: Automate mobilization

- Pick the three highest-frequency remediation actions in your environment (rotate credential, block IP, add security header, etc.). Automate them.
- Wire alerts to the channels operators actually read — Slack/Discord/SMS, not a dashboard nobody opens.
- Schedule a monthly CTEM review: how many cycles closed, what stages are slowest, what the next quarter's scope adjustment should be.

By day 90 you will not have a "perfect" CTEM program. You will have a working loop, which is infinitely more valuable than a perfect plan.

---

## 7. Metrics that matter

Most VM dashboards optimize for the wrong number. CTEM metrics should reflect the loop, not the queue.

- **Mean time to discover (MTTD).** From an exposure existing to your tooling knowing about it. Trend it down.
- **Mean time to validate (MTTV).** From discovery to a confirmed real-or-false-positive determination. Trend it down.
- **Mean time to mobilize (MTTM).** From validated finding to remediated and re-validated. Trend it down.
- **Exposure window for top-tier findings.** The interval during which a critical, validated, reachable exposure is open. This is the number that should keep you up at night.
- **Validation coverage.** Percentage of "open" findings that have been actively validated in the last 30 days. Anything below 50% means the queue is fiction.
- **Cycle close rate.** Number of CTEM cycles completed per quarter on a given scope.

Notably absent: total finding count, patch percentage, CVSS-weighted backlog. Those numbers reward activity, not outcomes.

---

## 8. Common failure modes

After watching dozens of teams attempt this, a few patterns repeat:

- **Scope creep on day one.** "Everything is in scope" means nothing is in scope. Start narrow.
- **Discovery without normalization.** Adding a sixth discovery tool without a single exposure store just multiplies noise.
- **Prioritization by committee.** Every stakeholder wants their pet finding on top. Codify the formula and let it run.
- **Validation by dashboard.** If the only validation is "the green light is on," the program will quietly rot.
- **Mobilization without owners.** Automated remediation is great; humans still need to own classes of risk.
- **Tooling without a loop.** Buying CTEM-branded products does not give you CTEM. Running the loop does.

---

## 9. How CTEM relates to SIEM, EDR, and SOC

A common point of confusion: if I have a SIEM, an EDR, and a SOC, do I still need CTEM? And vice versa — if I run a CTEM program, do I still need detection and response?

The answer is yes to both, because they operate on different sides of the incident timeline.

| Layer | Question it answers | When it acts |
|---|---|---|
| CTEM | Where could an attacker plausibly get in today? | **Before** anything happens. |
| SIEM | Across all my logs, is anything suspicious correlating right now? | **During** suspicious activity. |
| EDR | On this individual endpoint, is something malicious executing right now? | **During** an attack on a host. |
| SOC | What just tripped, and what should we do about it? | **During and after** an incident. |

A useful one-line mental model:

> **CTEM** = find and reduce exposures before they become incidents. **SIEM / EDR / SOC** = detect and respond when suspicious activity or attacks are happening.

### 9.1 What each layer does, in one sentence

- **SIEM** (Security Information and Event Management): the central log brain. Pulls in events from servers, apps, identity, cloud, and network gear and correlates patterns — for example, a user fails 50 logins, then succeeds from a new country.

- **EDR** (Endpoint Detection and Response): the security camera plus kill switch on each machine. Watches processes, files, network connections, command execution; can kill processes, quarantine devices, and roll back malicious changes.
- **SOC** (Security Operations Center): the people and processes that monitor those tools, investigate the alerts, and run incident response. Internal team, outsourced MDR, or some hybrid.

### 9.2 Why the layers need each other

A CTEM program with no detection layer is a list of risks that you may or may not catch when someone exploits one. A detection-and-response stack with no CTEM is a perpetual game of catch-up against exposures you could have closed in advance. The two layers feed each other:

- CTEM findings sharpen detection rules — if you know an exposure is reachable, the SIEM should treat traffic to it differently.
- SOC incidents sharpen CTEM scope — if the same class of exposure keeps producing incidents, that class moves up the prioritization queue.
- EDR telemetry validates CTEM assumptions — if reachability tests say a host is unreachable but EDR sees it taking inbound shell commands, your map is wrong.

### 9.3 Where ThreatCrush sits

ThreatCrush is built to operate in both layers from a single agent:

- **CTEM-side capabilities:** code scanner, pentest engine, marketplace ASM modules, exposure normalization, prioritization, validation re-runs.
- **Detect-and-respond capabilities:** inbound monitoring on every port and protocol (SIEM-style log + event collection), the daemon as an EDR-style on-host agent, real-time alerts (SOC-style notification), and active-defense modules (tar pits, honeypots, deception, automated abuse reports) for EDR-style response.

This is deliberate. Most teams under 200 engineers cannot afford the full nine-tool stack, and bolting CTEM onto an existing SIEM+EDR is its own integration project. ThreatCrush gives small and mid-sized teams a single substrate that does the obvious detect-and-respond work and runs the CTEM loop on top, with marketplace modules filling in the long tail.

It does **not** replace a mature enterprise SIEM or EDR — it coexists with them, feeding telemetry up and pulling exposure context down.

---

## 10. The open standards a serious platform should speak

CTEM is one taxonomy. The detection-and-response side of the world has its own — older, deeper, and largely open. A platform that takes itself seriously aligns with these standards rather than reinventing them, because it lets your team, your tooling, and your peers all use the same vocabulary.

The short version of the landscape:

| Standard | What it is | Where it fits |
|---|---|---|
| **MITRE ATT&CK** | Knowledge base of adversary tactics, techniques, and procedures. | Universal language for SIEM/EDR/SOC. |
| **MITRE D3FEND** | Knowledge graph of defensive countermeasures, mapped against ATT&CK. | The "what to do about it" side. |
| **Sigma** | Portable, vendor-neutral format for SIEM detection rules. | Detection rules that travel. |
| **YARA** | Pattern-matching language for files, malware families, and binary content. | File and payload detection. |
| **osquery** | SQL interface to endpoint state — processes, sockets, packages, users. | Open endpoint telemetry / EDR-style queries. |
| **OCSF / ECS / OSSEM** | Open schemas for normalizing security events. | Making logs from twelve sources look like one source. |
| **CTEM (ctem.org)** | Open taxonomy of exposures and exposure-management vocabulary. | The CTEM side of the stack. |
| **NIST CSF** | Five-function framework: Identify, Protect, Detect, Respond, Recover. | Governance, maturity, executive reporting. |
| **CIS Controls** | Concrete, prioritized list of practical security controls. | The "what should we actually do" checklist. |

### 10.1 Why this matters more than logos

It is easy to slap a row of standards logos on a marketing page. The harder, more useful question is: **what does each standard actually contribute to a working platform?**

- **ATT&CK** gives every alert and finding a stable identifier (e.g. `T1003.001 – LSASS Memory`). That identifier travels: into your SIEM, into your SOC's runbooks, into the EDR's response, into your

post-incident report. A platform that doesn't tag findings with ATT&CK technique IDs forces operators to translate.

- **D3FEND** gives the response side the same treatment. If ATT&CK tells you what an attacker did, D3FEND tells you which defensive techniques counter it. That mapping is what turns a detection into a playbook.
- **Sigma** gives detection rules portability. Modules that ship Sigma rules can be consumed by any SIEM-style processor — yours, ours, or someone else's. No vendor lock-in.
- **YARA** does the same for content patterns. A malware family signature in YARA is reusable across scanners, EDRs, and incident response tools.
- **osquery** gives you a uniform way to ask any host "what is going on right now?" — without writing a custom collector. ThreatCrush's daemon can dispatch osquery questions to itself and feed the answers back into the loop.
- **OCSF / ECS / OSSEM** are the schemas that keep your event store sane. Without them, every source is a dialect, and correlation becomes glue code.
- **NIST CSF** and **CIS Controls** are the language your CISO and your auditor speak. A platform that maps cleanly to them shortens your compliance conversations.

### 10.2 The minimum viable open-standards stack

For a real-time detection-and-response layer, the core foundation is roughly:

> **ATT&CK + Sigma + D3FEND** for the detection-and-response vocabulary, with **OCSF / ECS** to keep the event store coherent, and **NIST CSF + CIS Controls** for governance.

Add **YARA** when you care about file/payload detection, and **osquery** when you want open endpoint telemetry without a heavy proprietary EDR.

### 10.3 How ThreatCrush aligns

ThreatCrush is built so the open vocabulary is the default, not an afterthought:

- **Findings and detections carry ATT&CK technique IDs** wherever the underlying signal supports them. Your SOC sees `T1110.003 - Password Spraying`, not `Module 47 alert`.
- **Response modules reference D3FEND techniques** so that the action ThreatCrush takes (tar pit, kill, isolate, rotate) maps to a documented countermeasure family.
- **Detection logic ships as Sigma rules where applicable**, so a rule written for ThreatCrush can be exported to a corporate SIEM, and a Sigma rule written for a SIEM can be imported as a ThreatCrush module.
- **Events are emitted in an OCSF-compatible shape** (with ECS field aliases) so that a downstream SIEM, data lake, or BI tool can consume them without bespoke parsing.

- **Scope and control mappings reference NIST CSF functions and CIS Controls** so that the same event can roll up into a CTEM dashboard, a SOC investigation, and a compliance report without three different translations.

The point is not to claim ThreatCrush "supports" these standards. The point is that the standards are the substrate the platform is built on — so your team, your auditors, and your peers can read its output without a glossary.

---

## 11. Where ThreatCrush fits in your stack

ThreatCrush is not a replacement for everything. It is a replacement for the parts that no longer earn their keep:

- The legacy network IDS that only sees north-south traffic on port 80.
- The web vulnerability scanner that produces a PDF nobody reads.
- The "pentest as a service" subscription that runs once a year.
- The seven Python scripts your senior engineer wrote to glue the above together.

ThreatCrush replaces those with a single agent, a marketplace of modules, and an operator-first set of clients. It coexists happily with EDR, cloud-native security tools, SIEM, and SOAR — and it is designed to be the substrate on which a small team can run a real CTEM loop without needing a SOC of twelve people.

---

## 12. Next steps

If this resonates and you want to try it:

- Install the agent on a non-production server: `curl -fsSL https://threatcrush.com/install.sh | sh`
- Browse the Module Store to see what discovery, validation, and mobilization modules already exist.
- Read the Module Store HTTP API reference if you plan to publish your own.
- For air-gapped, FedRAMP-ready, or on-prem hardware appliance deployments, talk to us directly.

If you want to stay in the loop without committing to a deployment, join the waitlist — we send the next-stage releases and major module updates to the list first.

---

### About ThreatCrush

ThreatCrush is an all-in-one threat exposure platform for Linux servers, plus a marketplace of security modules. It is built and maintained by Profullstack and a community of module authors. The core platform is licensed for lifetime use; AI-enhanced modules are billed on usage.

Questions, feedback, or war stories: **hello@threatcrush.com**.

— *The ThreatCrush team*

# Appendix A. CTEM Readiness Checklist

*Do you actually run the loop, or do you run a queue?*

A CTEM program is five stages that repeat forever: scope, discover, prioritize, validate, mobilize. Most teams do two of them well and call it a program. Tick every control you can honestly say is running today — not planned, not purchased, running. The score at the end tells you which stage is starving the rest.

## 01 Scope

*Do you know what you are actually protecting?*

- [ ] **Scope is defined by business outcomes, not tool coverage**
  You can name the top five things an attacker could do that would genuinely hurt the business, and trace each one back to the systems and identities that would have to fall first. If your scope is "everything the EDR sees," a vendor renewal can silently halve it.
- [ ] **A live asset inventory exists and is not a spreadsheet**
  Servers, services, domains, cloud accounts, and third-party SaaS are enumerated automatically and refreshed. An inventory a human maintains by hand is out of date the week it is written.
- [ ] **Identities are in scope alongside machines**
  Dormant accounts, over-privileged service principals, and MFA gaps are tracked as exposures. Most modern intrusions arrive through a credential, not a CVE.
- [ ] **Every in-scope system has a named human owner**
  Not a team alias. When a validated finding lands, there is one person whose job it is to close it. Unowned findings are the ones that outlive the engineer who filed them.
- [ ] **Scope is revisited on a schedule, not after an incident**
  A quarterly review that adds and, crucially, removes things. Scope that only ever grows becomes scope nobody believes.

## 02 Discover

*Would you find out, or would someone tell you?*

- [ ] **External attack surface is enumerated continuously**
  Subdomains, exposed dashboards, forgotten storage buckets, and stale DNS records. The exposures that get exploited are usually the ones nobody remembered deploying.
- [ ] **Inbound traffic is monitored on every port, not just the ones you expect**
  Passive monitoring catches the service you forgot you exposed. Scanning only your declared surface finds only what you already knew about.
- [ ] **Code-side discovery runs on every commit**
  Hardcoded secrets, vulnerable and malicious dependencies, and dangerous configuration land in the same exposure store as infrastructure findings — before they reach production, not after.
- [ ] **Dependency and supply-chain risk is tracked, including transitive packages**
  Your build pulls code from people you have never met. A direct-dependency-only view misses the layer where compromises actually happen.
- [ ] **All discovery sources feed one normalized exposure store**
  The same finding from three tools is one row, not three. Adding a sixth discovery tool without normalization multiplies noise instead of coverage.
- [ ] **Leaked credentials are monitored outside your perimeter**
  Public repos, paste sites, and breach dumps. Learning about a breach when the stolen credential is used is late, but it beats not learning at all.

## 03 Prioritize

*Does your ranking survive contact with an attacker?*

- [ ] **A written prioritization formula exists and is applied mechanically**
  Inputs: exploitability, attacker reachability, blast radius, business impact, compensating controls. Written down, so it runs the same way whether or not the loudest stakeholder is in the room.
- [ ] **Ranking is not primarily CVSS**
  CVSS describes a vulnerability in the abstract. It knows nothing about whether the affected host is reachable, whether the service is running, or whether the data behind it matters.
- [ ] **Real-world exploitation data feeds the ranking**
  Known-exploited catalogues and exploit-prediction scoring separate the thousands of theoretical findings from the handful being used this week.
- [ ] **Reachability is evaluated, not assumed**
  A critical vulnerability in a code path that never executes, on a host no route reaches, is not a critical exposure. Treating it as one is how backlogs become fiction.
- [ ] **A published top-20 list exists with sign-off from engineering**
  Short enough to actually finish, owned by the people who will do the work, and refreshed each cycle.

## 04 Validate

*Do you confirm, or do you trust the dashboard?*

- [ ] **Every top-tier finding has a defined validation check**
  A script, a pentest module, or a control test that answers "is this real?" — scheduled, not run once during triage.
- [ ] **Tickets close on validated fixes, not on claimed fixes**
  "Patched" and "no longer exploitable" are different statements. Only one of them is evidence.
- [ ] **Controls are tested by trying to defeat them**
  The green light on a console means the agent is reporting, not that the control works. Attack your own infrastructure — safely, and only your own.
- [ ] **False positives are measured and fed back into tuning**
  A scanner nobody trusts is a scanner nobody reads. If you cannot state your false-positive rate, your team has already started ignoring the output.
- [ ] **Validation coverage is tracked as a metric**
  The share of open findings actively validated in the last 30 days. Below 50%, the queue is a work of imagination.

## 05 Mobilize

*Do fixes ship, or do they get assigned?*

- [ ] **Your three most frequent remediations are automated**
  Rotate the credential, block the address, add the header. The repetitive fixes should not consume senior engineering time.
- [ ] **Alerts route to channels operators actually read**
  Chat, SMS, or phone — not a dashboard someone opens on Mondays. An alert nobody sees is an outage nobody is fixing.
- [ ] **Findings become tickets in the system engineering already lives in**
  Work that requires logging into the security tool is work that competes with the sprint board and loses.
- [ ] **An incident response plan exists and has been rehearsed this year**
  Who to call, who decides, who talks to customers. A plan that has never survived a tabletop is a document, not a capability.
- [ ] **Backups are restored on a schedule, not merely taken**
  An untested backup is a hypothesis. Restore drills are the only thing that converts it into a recovery capability.
- [ ] **A recurring review asks which stage is slowest**
  Cycles closed, stage latency, next quarter's scope adjustment. Without it the loop degrades back into a queue and nobody notices for two quarters.

### Scoring — 27 controls in total

Count your ticks, divide by 27, multiply by 100.

**0–29% Reactive** — You have tools, not a loop. Findings arrive, get triaged by whoever is free, and the backlog only grows. Start at scoping — everything downstream is noise until scope is real.

**30–54% Emerging** — Discovery works and somebody owns the queue, but nothing closes the loop. Validation is almost certainly your bottleneck: you cannot tell a fixed finding from a stale one.

**55–79% Operating** — The loop turns. The work now is latency — shrinking the exposure window between discovery and re-validated fix, and automating the three remediations you do most often.

**80–100% Optimizing** — You run a real CTEM program. The remaining gains are in scope adjustment and cost imposition: making attacks expensive rather than merely detected.