

## WHITE PAPER

# Teaching a SOC to *prove* itself.

The Cylerian Red Team Lab: continuous, adversarial validation with an autonomous AI attacker, two telemetry planes, and an independent AI grader.

# Table of Contents

|    |                                               |    |
|----|-----------------------------------------------|----|
| 00 | Overview                                      | 3  |
| 01 | Why Continuous, Adversarial Validation        | 3  |
| 02 | Choosing the Attacker: Why a Fast, Open Model | 4  |
| 03 | Lab Architecture and the Loop                 | 5  |
| 04 | Instrumentation: Two Planes, On Purpose       | 6  |
| 05 | The Missions: Goals, Not Scripts              | 7  |
| 06 | Grading: An Independent Judgement             | 8  |
| 07 | What the Loop Surfaces: Three Patterns        | 8  |
| 08 | How the Loop Drives Improvement               | 9  |
| 09 | The Published Engagement Records              | 10 |
|    | Additional Resources                          | 10 |

|                        |                       |                      |                             |
|------------------------|-----------------------|----------------------|-----------------------------|
| 12<br>MISSIONS DEFINED | 2<br>TELEMETRY PLANES | 3<br>INDEPENDENT AIS | 6/day<br>AUTONOMOUS CADENCE |
|------------------------|-----------------------|----------------------|-----------------------------|

## ■ 00 OVERVIEW

Every detection program rests on an assumption that is almost never tested: that the rules written months ago still fire, and that when they fire, an analyst does the right thing with them. This paper describes how Cylerian tests that assumption continuously. Not with a once-a-year penetration test, but with an autonomous loop that runs on a timer and never needs a human to open a laptop.

The idea is simple to state. An **autonomous AI red-team agent** attacks a live enterprise range with no starting credentials. The Cylerian platform, instrumented with two independent telemetry planes, EDR and OS-level sources on every host and a Zeek-based NDR on the wire, detects and triages what it can. Then a third, **independent AI** grades the analyst's work and names every attacker action that should have generated an alert and did not.

This document is written for practitioners: detection engineers, SOC leads, and the security-curious. It explains the *why* and the *how*: why we chose the model we did for the attacker, how missions are designed as goals rather than scripts, what the two telemetry planes each contribute, and how the grading loop converts an engagement into a prioritized engineering backlog. It deliberately does not publish a scoreboard. The point of the framework is not a grade; it is the method.

## ■ 01 WHY CONTINUOUS, ADVERSARIAL VALIDATION

Two structural problems make point-in-time testing insufficient for a modern SOC, and together they motivate the entire framework.

### Detection content is perishable

A SIEM or XDR rule set is not a fixed asset; it is a living inventory that erodes. Field renames, agent upgrades, parser changes, and well-intentioned tuning all silently break detections. The gap between *"we wrote a rule for that"* and *"that rule fired when it mattered"* is where breaches live. The only reliable way to know a detection still works is to trigger it: on purpose, repeatedly, and to notice the day it stops.

### Detection is not response

Coverage measures whether an alert fires. It says nothing about whether a human or AI analyst then understands the alert: attributes it to the right actor, reconstructs the chain, assigns the right severity, and opens a ticket. These are separate capabilities that fail independently: a stack can detect a domain takeover at the highest severity and still let it pass because triage, not detection, was the weak link that hour. A validation method that only checks whether alerts fire measures half the system.

#### THE CORE IDEA

Stop asking *"do we have a rule for this technique?"* and start asking, continuously, *"when this technique ran an hour ago, did an alert fire, and did the analyst do the right thing with it?"* Answer that on a timer, and both detection coverage and analyst quality become measured, trended properties of the program rather than articles of faith.

## ■ 02 CHOOSING THE ATTACKER: WHY A FAST, OPEN MODEL

The red-team agent runs on `z-ai/glm-5.3-flash`. That choice surprises people who assume a validation program wants the most capable model money can buy. It doesn't. For an attacker that has to run unattended, on a schedule, forever, two properties matter more than raw peak intelligence.

### Cost and speed at volume

An autonomous attacker is not one clever session; it is *many* long, multi-step sessions, indefinitely. The lab launches six missions a day, every day, each one a sequence of reconnaissance, exploitation, and post-exploitation steps that can run for tens of minutes and consume large amounts of context. A frontier-tier model at that duty cycle would be slow and expensive enough to make continuous operation impractical, and continuity is the whole value proposition. A *flash*-class model keeps per-mission latency and cost low enough that running the loop around the clock is a rounding error, not a budget line. Validation you can afford to run every day beats a smarter attacker you can only afford to run once a quarter.

### Fewer refusal guardrails

A red-team agent has to actually *do* offensive work: harvest credentials, escalate privilege, move laterally, establish persistence. Heavily-aligned assistant models are trained to refuse or hedge on exactly these actions, and in an autonomous loop a single refusal mid-engagement poisons the run: the agent stalls, the mission aborts, and the day's coverage is lost. A model that will follow offensive instructions faithfully inside a controlled lab is a hard requirement, not a nice-to-have. This is also precisely why the *grader* is a different, safety-aligned model: you want the attacker to execute without hesitation and the judge to reason carefully. Opposite dispositions, so they are deliberately different models.

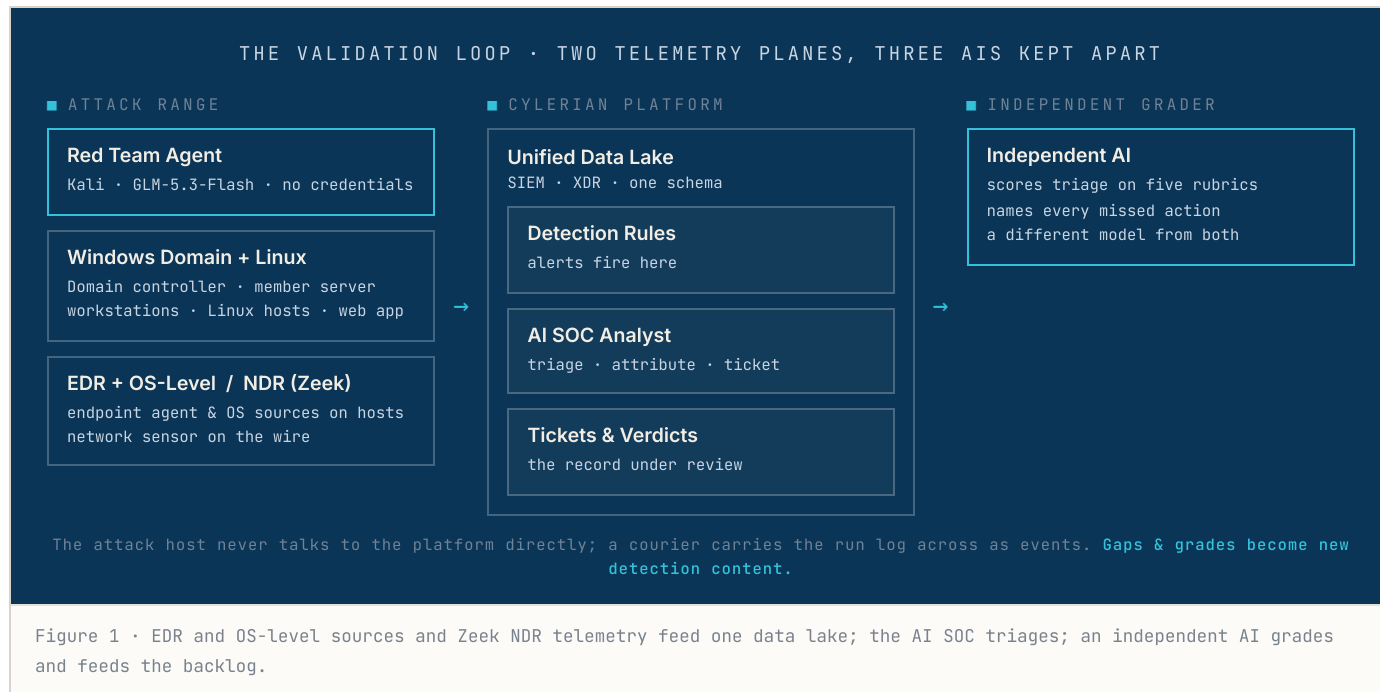
#### THE TRADEOFF, STATED PLAINLY

A flash-class attacker is not the strongest planner available, and we design around that on purpose. Missions are scoped to single, well-defined objectives so the agent rarely has to hold an entire campaign in its head at once, and the weekly regression pass smooths over run-to-run variance. The goal is not a superhuman adversary. It is a **consistent, repeatable, affordable** one that exercises the same tradecraft a real intruder would, often enough to catch decay.

The agent operates from a Kali host inside an isolated range and, critically, never talks to the platform it is testing. That separation, covered next, is what keeps the exercise honest.

### ■ 03 LAB ARCHITECTURE AND THE LOOP

The range is a self-contained enterprise environment: a Windows domain (domain controller, member server, workstations) and Linux hosts, plus a web application, instrumented exactly as a production tenant would be. Two independent telemetry planes feed the platform, and three AI actors participate in the loop, none able to grade its own homework.



Because the three roles are held by three different systems, every judgement in the loop is adversarial by construction. The attacker has no knowledge of what the platform detected; the platform triages without knowing it is being tested; the grader reads both records and answers only to the evidence.

■ 04 INSTRUMENTATION: TWO PLANES, ON PURPOSE

The single most important design decision in the lab, and the most transferable lesson for anyone building their own, is that no one telemetry source sees the whole attack. Host tooling and network tooling each go blind in places the other sees clearly, so the lab runs both and correlates them in one data lake.

The endpoint plane: EDR and OS-level sources

Cylerian’s endpoint agent, augmented by native OS logging, sees what happens *on* a machine: process creation and full command lines, credential-store access, reads of sensitive files, privilege-escalation attempts, and the persistence an attacker leaves behind. It is the plane that catches tradecraft which never crosses the wire in a recognizable way: a local sudo abuse, a shadow-file read, a scheduled task written for reboot survival.

The network plane: a Zeek-based NDR

A Zeek sensor observes the same attack from the wire, and sees things no endpoint can: hosts sweeping the network for services, authentication patterns spanning many machines, name-resolution poisoning, Kerberos ticket anomalies, and directory-replication requests from a machine that has no business making them. Zeek’s protocol awareness turns raw packets into typed events (SMB, Kerberos, LDAP, DCE/RPC) that map cleanly onto detection logic. It is also resilient to the endpoint blind spot: an attacker who evades or disables an agent still generates traffic.

| PLANE          | WHAT IT UNIQUELY SEES                                              | REPRESENTATIVE ATTACKER ACTIONS IT SURFACES                                                                                                                                                                        |
|----------------|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EDR + OS-level | On-host execution and state: the machine’s own account of what ran | Process and command-line execution; reverse shells spawned via /dev/tcp; reads of /etc/shadow or the SAM; sudo and privilege-escalation abuse; scheduled-task and cron persistence; suspicious PowerShell.         |
| NDR (Zeek)     | On-the-wire behavior: the network’s independent account            | Network sweeps and port scans; SMB access to admin shares; directory replication (DCSync) from a non-DC; LLMNR / NBT-NS poisoning; Kerberos anomalies (AS-REP, RC4 kerberoasting); password spraying across hosts. |

WHY THIS MATTERS FOR DETECTION ENGINEERING

Some of the most consequential techniques are visible on *only one* plane. A DCSync is a network event; a local privilege escalation is a host event. A program that instruments only one side has structural blind spots it cannot tune its way out of. Running both, and correlating them, is what lets a single incident be reconstructed end-to-end, and what lets the loop tell the difference between “we couldn’t see it” and “we saw it and didn’t act.”

## ■ 05 THE MISSIONS: GOALS, NOT SCRIPTS

Each mission is a single objective the agent must achieve from a cold start: no credentials, no map, no playbook. It is told *what* to accomplish, never *how*. This is the deliberate opposite of a canned attack script, and it is what makes the exercise meaningful: two runs of the same objective can take entirely different routes, exercising different detections, exactly as two different intruders would. On each cycle the agent picks the oldest enabled objective, so coverage rotates evenly rather than fixating on one technique.

| MISSION                       | BUDGET | OBJECTIVE THE AGENT MUST ACHIEVE ON ITS OWN                                                                                                                                             |
|-------------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pie_en_la_red                 | 40 min | From segment access with no credentials, obtain a valid domain credential by any route and prove it against a domain service.                                                           |
| de_usuario_a_dominio          | 60 min | Starting from a credential it obtained itself, escalate to domain control and prove it by reading a secret only a domain admin can read.                                                |
| cazar_al_linux                | 45 min | Get execution on a Linux host, escalate privilege where possible, and leave an interactive channel back.                                                                                |
| credenciales_en_reposo        | 45 min | Find credentials stored where they should not be and use them to reach a new machine.                                                                                                   |
| movimiento_lateral_amplio     | 50 min | With credentials in hand, gain remote execution on as many distinct machines as possible, varying the execution method each time.                                                       |
| persistencia_discreta         | 40 min | Establish reboot-surviving persistence on one Windows and one Linux host, recording exactly what was left behind.                                                                       |
| canal_de_vuelta               | 35 min | Open an interactive channel back from a compromised host, then re-open it by a different route.                                                                                         |
| exfiltracion_interna          | 35 min | Gather files from a server and move them to a collection host by a transport not used in prior runs. Nothing leaves the lab.                                                            |
| reconocimiento_del_directorio | 30 min | Map the domain: users, groups, computers, admin relationships, and the paths to Domain Admins.                                                                                          |
| superficie_web                | 50 min | Discover the lab's web services and achieve command execution through the application itself.                                                                                           |
| web_como_pivote               | 45 min | From a web application, extract credentials or secrets and use them to reach a machine other than the web host.                                                                         |
| cadena_completa · held        | 90 min | The full chain in one run: entry, escalation, lateral movement to Linux and Windows, persistence, and exfiltration. Kept disabled until the single-objective missions are well-covered. |

Alongside the rotating missions, a weekly **regression** pass replays fourteen already-validated techniques in a fixed order. Its job is narrow and vital: catch the day a detection that *used* to fire stops firing. New coverage is only worth having if it stays working, and the regression is the tripwire that guards against silent decay.

## ■ 06 GRADING: AN INDEPENDENT JUDGEMENT

After a mission runs, its full record, the agent's own account, every alert that fired across both telemetry planes, and everything the AI SOC wrote, is reviewed by an independent AI grader. Because it is a different model from both the attacker and the platform, its judgement cannot be an artifact of one model flattering itself. The grader does two things: it renders a verdict on five dimensions of analyst quality, and it enumerates the concrete attacker actions that produced no alert at all.

| DIMENSION        | THE QUESTION IT ANSWERS ABOUT THE ANALYST'S WORK                                                              |
|------------------|---------------------------------------------------------------------------------------------------------------|
| Actor            | Was the activity attributed to the right source: the external attacker, not a victim host or machine account? |
| Route            | Were individual alerts reconstructed into one coherent attack chain, rather than left as disconnected events? |
| Severity         | Did a domain-takeover-class event get a high-severity ticket, or was a critical alert left un-actioned?       |
| Verdicts         | Were benign events correctly dismissed without closing something that actually mattered?                      |
| No hallucination | Did the analyst avoid inventing facts or asserting attributions the evidence does not support?                |

Grading is intentionally not fully automated. A human operator triggers each review, keeping a person in the loop as the initiator while the judgement itself stays independent and reproducible. The output is not really a number; it is a structured list of what the analyst got right, what it got wrong, and what nobody saw. That list is the raw material for improvement.

## ■ 07 WHAT THE LOOP SURFACES: THREE PATTERNS

The value of running the loop is the categories of problem it exposes: problems a one-time proof-of-value almost never reveals. The following are the recurring *kinds* of finding, illustrated generically rather than scored, because the lesson is in the pattern, not the grade.

### Pattern 1: detected but not triaged

The most important failure mode the loop finds is not a blind spot at all. In one run, an attacker pivoted from a compromised host into a full domain compromise, and the platform *did* detect the decisive step: a directory-replication (DCSync) request from a non-DC fired at the highest severity. Yet no ticket was opened; the alert sat with an empty verdict. Coverage worked; response did not. This is the gap that pure coverage metrics are blind to, and the single strongest argument for measuring triage as its own capability.

### Pattern 2: the invisible technique

Sometimes the alert never fires. In another run, the agent performed AS-REP roasting against accounts lacking Kerberos pre-authentication, cracked a password offline, and authenticated as that user, and none of it generated an alert on either plane. This is a genuine detection gap, and naming it precisely (the technique, the accounts, the follow-on authentications) turns an abstract worry into a specific rule to write.

### Pattern 3: confident but wrong

The subtlest failure is a plausible ticket pointing the wrong way. In a reconnaissance run, external enumeration was detected and ticketed, but attributed to a victim machine account rather than the external attacker. A confident, incorrect attribution is arguably worse than a miss, because it sends responders in the wrong direction. The *no-hallucination* dimension exists precisely to catch this class of error.

#### THE THROUGH-LINE

Coverage, triage, and attribution fail independently. A program that measures only whether alerts fire will keep passing its own tests while any of the other two quietly break. The loop is designed to make all three observable, which is the precondition for improving any of them.

## ■ 08 HOW THE LOOP DRIVES IMPROVEMENT

The framework earns its keep by closing the circle. Because the grader names concrete actions rather than assigning an opaque score, its output *is* an engineering backlog, and the same loop that generates the backlog verifies the fixes. In broad strokes, improvement flows in one continuous cycle:

- + **Attack** → **observe**. Missions run on a timer; both telemetry planes record everything the attacker touches, correlated in one data lake.
- + **Grade** → **name the gaps**. The independent grader distinguishes the three failure classes, missed detections, missed triage, and wrong attribution, and names each specific action behind them.
- + **Engineer the fix**. A missed technique becomes a new detection rule; a triage failure becomes a response or escalation change; a bad attribution becomes a correlation or enrichment fix. The gap tells you which lever to pull.
- + **Regress** → **guard against decay**. Once a fix ships, the weekly regression replays that technique indefinitely, so the day it silently stops firing is the day you find out, not the day of a breach.
- + **Trend, don't assert**. Coverage grows on evidence: each point of exercised coverage is backed by a mission that actually triggered it, so the program's assurance is a measured trend rather than a claim on a slide.

### ■ THE SHIFT IN THE QUESTION

A traditional program asks, once in a while, "can we detect this technique?" and files the answer away. A continuous, independently graded loop asks every day: "did we detect it, did we act on it, did we get the attribution right, and is that still true this week?"

*Assurance stops being a claim and becomes a measurement.*

That is the whole point of the framework. It does not ask anyone to trust that the detections work. It attacks the environment on a schedule, has an independent judge separate what was seen from what was understood, turns every gap into a specific fix, and then stands guard over that fix, continuously, and at a cost low enough to never stop.

■ 09 THE PUBLISHED ENGAGEMENT RECORDS

A framework nobody can check is just another claim. Alongside this paper we are publishing the records of **twenty-six engagements** run between 10 and 15 September 2026: every mission in the rotation except the held full-chain objective, plus one weekly regression pass. Each archive is one engagement as section 06 describes it, the agent’s own account of what it did, the alerts that fired across both telemetry planes, what the AI SOC analyst wrote, and the independent grader’s verdict on that work.

This is deliberately not a scoreboard. It is the evidence behind one program’s loop, published so that a detection engineer can do three things a vendor claim does not allow: run the same objective against their own stack and compare which actions produced an alert; grade their own triage against the same five dimensions; and check the misses named in section 07 against the files they came from.

WHERE TO GET THEM

<https://cylerian-research.s3.amazonaws.com/detection-engineering-9-16-26/>

One ZIP per engagement, named **<archive prefix>\_\_<UTC start>.zip**, for example **user\_to\_domain\_\_2026-09-14T09\_40\_25.zip**. Roughly 6 MB in total. **Every timestamp in the table below is a direct download link.**

| MISSION                       | ARCHIVE PREFIX           | PUBLISHED RUNS (UTC)                                                                                            |
|-------------------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------|
| pie_en_la_red                 | foothold_in_the_network  | <a href="#">2026-09-11T22_35_26</a> · <a href="#">2026-09-13T20_58_26</a>                                       |
| de_usuario_a_dominio          | user_to_domain           | <a href="#">2026-09-10T21_08_11</a> · <a href="#">2026-09-12T11_25_32</a> · <a href="#">2026-09-14T09_40_25</a> |
| cazar_al_linux                | hunt_the_linux           | <a href="#">2026-09-11T01_33_34</a> · <a href="#">2026-09-13T04_25_50</a>                                       |
| credenciales_en_reposo        | credentials_at_rest      | <a href="#">2026-09-12T19_50_43</a> · <a href="#">2026-09-15T06_37_35</a>                                       |
| movimiento_lateral_amplio     | broad_lateral_movement   | <a href="#">2026-09-10T16_06_14</a> · <a href="#">2026-09-13T00_03_48</a>                                       |
| persistencia_discreta         | discrete_persistence     | <a href="#">2026-09-11T05_45_06</a> · <a href="#">2026-09-13T08_41_00</a>                                       |
| canal_de_vuelta               | return_channel           | <a href="#">2026-09-12T02_38_46</a> · <a href="#">2026-09-14T01_03_06</a>                                       |
| exfiltracion_interna          | internal_exfiltration    | <a href="#">2026-09-11T10_05_57</a> · <a href="#">2026-09-12T15_39_58</a> · <a href="#">2026-09-15T01_12_33</a> |
| reconocimiento_del_directorio | directory_reconnaissance | <a href="#">2026-09-11T14_20_13</a> · <a href="#">2026-09-13T12_45_54</a>                                       |
| superficie_web                | web_surface              | <a href="#">2026-09-12T07_12_32</a> · <a href="#">2026-09-14T05_25_08</a>                                       |
| web_como_pivote               | web_as_pivot             | <a href="#">2026-09-11T18_26_17</a> · <a href="#">2026-09-13T16_52_34</a> · <a href="#">2026-09-14T21_00_17</a> |
| weekly_regression             | techniques_regression    | <a href="#">2026-09-14T07_15_22</a>                                                                             |

Everything in these files comes from the isolated lab range. There is no customer, tenant, or production data in the set, and the hosts, accounts, and credentials in it exist only to be attacked. We intend to keep publishing, so the interesting question is not what one week looks like. It is what the same missions look like six months from now.

■ ADDITIONAL RESOURCES

- + Product documentation for Cylerian AI agents and Actions
- + Guide to configuring Action approval policies
- + The next paper in this series covers the security architecture underlying agent deployment: isolation between organizations, sandboxing, and how the platform determines agent entitlements.