CPF SOC Integration Guide: Mapping the 100 Indicators to SIEM Platforms and Data Sources

A Technical Reference

Giuseppe Canale, CISSP
Independent Researcher
g.canale@cpf3.org
cpf3.org
ORCID: 0009-0007-3263-6897

February 2026

Abstract

The CPF defines 100 pre-cognitive vulnerability indicators across 10 categories. The Implementation Companion [2] provides the detection logic. This guide provides the plumbing: how to connect that logic to the tools a Security Operations Center already uses.

The document operates at two levels. At the data source level, it maps CrowdStrike Falcon, Qualys VMDR, Tenable.io, and Rapid7 InsightVM to the specific CPF indicator categories they feed. At the SIEM level, it provides query templates in SPL (Splunk), KQL (Elastic and Microsoft Sentinel), and AQL (QRadar) that extract the signals the CPF engine requires. Each section is self-contained: an engineer working with Splunk needs only the Splunk section, regardless of which data sources are in use.

No new tools are required. The CPF engine runs as an additive layer [4], consuming data that the SOC already collects.

1 Introduction

This guide assumes the reader has access to the Implementation Companion [2], which defines the 100 indicators, the OFTLISRV detection schema, and the Convergence Index calculation. This guide does not restate that logic. It answers a single question: given the indicators the Companion defines, where does the data come from, and how do you extract it from the tools you already have?

The guide is organized around two distinct integration layers. The first layer connects vulnerability scanners and endpoint detection tools to the CPF data pipeline. These tools produce raw signals—patch histories, alert volumes, endpoint behavioral data—that the CPF engine transforms into indicator states. The second layer connects the CPF engine to the SIEM platform where correlation happens. This is where queries live, where the Convergence Index is computed, and where alerts are generated.

1.1 What This Guide Does Not Cover

This guide does not cover the detection logic itself—that is the Companion's job. It does not cover validation methodology—that is the Validation Roadmap [3]. It does not cover the deployment strategy—that is the Deployment-Validation Loop [4]. It covers only the technical plumbing between CPF and the SOC's existing infrastructure.

2 Architecture Overview

The integration architecture has three layers. Data sources produce raw telemetry. The SIEM ingests, normalizes, and correlates that telemetry. The CPF engine, running as a query layer within the SIEM, reads the normalized data and computes indicator states.

Table 1: Integration architecture layers

Layer	**Role**	**Tools**
Data Sources	Raw telemetry	CrowdStrike, Qualys, Tenable, Rapid7
SIEM	Ingest & correlate	Splunk, Elastic, Sentinel, QRadar
CPF Engine	Indicator comp.	Queries within SIEM
Output	Alerts & dash.	Native SIEM alerting

The CPF engine does not run as a separate service. It runs as a set of scheduled queries and correlation rules within the SIEM. This is by design: it keeps the deployment additive and eliminates the need for a separate data pipeline.

2.1 Indicator-to-Data-Source Mapping

Not every data source feeds every indicator category. The mapping below shows which categories each source contributes to. An organization does not need all four data sources—even a single vulnerability scanner provides enough signal for categories [2.x], [5.x], and [10.x].

Table 2: Which data sources feed which indicator categories

Category	**CS**	**Qual**	**Ten**	**R7**	**SIEM**
[1.x] Authority	✓				
[2.x] Temporal		✓	✓	✓	
[3.x] Social	✓				
[4.x] Affective					✓
[5.x] Cognitive	✓	✓	✓	✓	
[6.x] Group					✓
[7.x] Stress	✓	✓	✓	✓	
[8.x] Unconscious					∼
[9.x] AI-Specific					✓
[10.x] Convergent	✓	✓	✓	✓	

CS=CrowdStrike. Qual=Qualys. Ten=Tenable. R7=Rapid7. SIEM=native SIEM data.

∼ = retrospective only. See Section 9 for details.

3 Data Source Integration

This section covers how each data source connects to the SIEM and what fields the CPF engine expects. All data sources connect via their native SIEM integration modules—no custom connectors are needed.

3.1 CrowdStrike Falcon

CrowdStrike is the primary source for endpoint behavioral data. The CPF engine uses three data streams from Falcon.

Detections. Each detection event carries a severity, a tactic classification, and a response action (blocked, allowed, or quarantined). The CPF engine uses the ratio of blocked-to-allowed detections over time to compute alert fatigue signals for category [5.x]. The relevant fields after SIEM ingestion are: `detection.severity`, `detection.tactic`, `response.action`, and `event.timestamp`.

Process telemetry. Process creation events with parent-child relationships feed category [1.x] (authority). A process launched by an administrative tool that in turn spawns a shell—without matching an approved workflow—is a signal that authority-based override patterns may be active. Fields: `process.name`, `process.parent.name`, `user.roles`.

Identity protection events. When Falcon Identity Protection is enabled, lateral movement attempts and privilege escalation events feed categories [3.x] (social influence) and [7.x] (stress response). Fields: `identity.action`, `identity.source_user`, `identity.target_user`.

3.2 Qualys VMDR

Qualys feeds vulnerability lifecycle data: when a vulnerability was first detected, when it was last confirmed, and whether it has been patched. This is the primary signal for temporal indicators [2.x].

The CPF engine requires these fields after SIEM ingestion: `vuln.cve`, `vuln.first_detected`, `vuln.last_confirmed`, `vuln.status`, `host.name`, `vuln.severity`.

Qualys also provides `vuln.patch_status`, which distinguishes patched from unpatched hosts. This is the signal the Companion uses for repetition compulsion detection in [2.x].

3.3 Tenable.io

Tenable provides similar vulnerability lifecycle data to Qualys, with one addition relevant to CPF: the `vuln.vpr` field incorporates exploitation likelihood. The CPF engine uses VPR alongside CVSS to detect splitting: if an organization patches high-VPR vulnerabilities on some systems but not others, that disparity is a [2.x] signal.

Required fields: `vuln.cve`, `vuln.vpr`, `vuln.cvss`, `vuln.first_seen`, `vuln.last_seen`, `vuln.status`, `asset.id`.

3.4 Rapid7 InsightVM

Rapid7 provides vulnerability data with a strong asset-risk scoring model. The CPF engine uses `asset.risk` to detect cognitive overload [5.x]: when aggregate risk is high but patch rate is low, the Companion's alert fatigue formula applies directly.

Required fields: `vuln.cve`, `asset.id`, `asset.risk`, `vuln.first_found`, `vuln.status`, `vuln.severity`.

4 SIEM Integration: Splunk

Splunk is the most common SIEM in enterprise SOCs. The CPF engine runs as a set of SPL searches and correlation searches within Splunk.

4.1 Prerequisite: Data Normalization

All CPF queries assume data has been normalized to Splunk's Common Information Model (CIM). If your data source integrations use CIM-compatible Splunk Add-Ons (which all four supported sources provide), no additional normalization is needed. The queries below use CIM field names.

4.2 Temporal Vulnerability Detection [2.x]

This search identifies vulnerabilities open longer than 90 days—the primary temporal vulnerability signal.

```
index=vulnerability_scan
  status="Active"
| eval days_open =
    (now() - strtotime(first_detected)) / 86400
| where days_open > 90
| stats count as open_count,
    avg(days_open) as avg_days
    by host, vulnerability.cve, severity
| sort - avg_days
| eval cpf_signal =
    if(avg_days > 180, "RED",
    if(avg_days > 90, "YELLOW", "GREEN"))
```

Listing 1: SPL: Temporal vulnerability detection

4.3 Alert Fatigue Detection [5.x]

This search computes the investigation rate decay over time—the core signal for cognitive overload indicators.

```
index=crowdstrike_detections
| eval week = strftime(
    _time, "%Y-%W")
| stats count as total_alerts,
    count(eval(response_action="investigated"))
      as investigated
    by week, analyst
| eval invest_rate =
    investigated / total_alerts
| join type=left analyst [
    search index=crowdstrike_detections
    | eval week = strftime(_time, "%Y-%W")
    | stats count as t_total,
        count(eval(response_action="investigated"))
          as t_inv by week, analyst
    | eval rate = t_inv / t_total
    | stats max(rate) as baseline_rate
      by analyst
  ]
| eval fatigue_score =
    1 - (invest_rate / baseline_rate)
| eval cpf_signal =
    if(fatigue_score > 0.7, "RED",
    if(fatigue_score > 0.4, "YELLOW", "GREEN"))
```

Listing 2: SPL: Alert fatigue curve

4.4 Authority Override Detection [1.x]

This search identifies processes launched by administrative accounts that deviate from approved baselines—the signal for authority-based vulnerability indicators.

```
index=crowdstrike_process
  user.roles="Administrator"
| eval parent_admin =
    if(process.parent.name IN (
      "svchost.exe","explorer.exe",
      "cmd.exe","powershell.exe"), 1, 0)
| eval shell_spawn =
    if(process.name IN (
      "cmd.exe","powershell.exe",
    "bash.exe","sh.exe"), 1, 0)
| where parent_admin=1 AND shell_spawn=1
| bin _time as hour
| stats count as override_count
    by host, user.name, hour
| eval cpf_signal =
    if(override_count > 5, "RED",
    if(override_count > 2, "YELLOW", "GREEN"))
```

Listing 3: SPL: Authority override detection

4.5 Convergence Index Computation

The Convergence Index aggregates all active indicator states into a single organizational risk score. The query below uses equal weights for illustration. In production, replace the numerator with a weighted sum using the w_i values from the Companion's Bayesian network (Section 8 defines the full weighted formula). In Splunk, this runs as a scheduled search that reads the outputs of the individual indicator searches.

```
index=cpf_indicator_states
| stats
    count(eval(cpf_signal="RED")) as red,
    count(eval(cpf_signal="YELLOW")) as yellow,
    count(eval(cpf_signal="GREEN")) as green
| eval total = red + yellow + green
| eval CI =
    (red * 1.0 + yellow * 0.5) / total
| eval CI_state =
    if(CI > 0.6, "CRITICAL",
    if(CI > 0.4, "HIGH",
    if(CI > 0.2, "ELEVATED", "NORMAL")))
| table CI, CI_state, red, yellow, green
```

Listing 4: SPL: Convergence Index

5 SIEM Integration: Elastic

Elastic uses the Elastic Common Schema (ECS) for field normalization. The CPF engine runs as a combination of saved searches and detection rules within Elastic Security.

5.1 Prerequisite: ECS Mapping

CrowdStrike, Qualys, Tenable, and Rapid7 all have official Elastic integrations that map to ECS. Verify that your integrations are using ECS-compatible mappings before deploying CPF queries. Key ECS fields used throughout: `event.category`, `event.action`, `host.name`, `user.name`, `process.name`, `process.parent.name`.

5.2 Temporal Vulnerability Detection [2.x]

```
event.category:"vulnerability" AND
  vulnerability.status:"open" AND
  NOT vulnerability.patched:true AND
  vulnerability.first_detected:<now-90d
```

Listing 5: KQL: Temporal vulnerability filter

The KQL filter above selects the raw event set. Elastic does not support aggregation in KQL directly—use an **ML job** for the full detection. Train on 30 days of `days_open` grouped by `host.name`. Anomalies exceeding the host's historical average by more than 2 standard deviations trigger a CPF [2.x] signal.

Alternatively, implement as a **Transform job** that reads the filtered events, computes `days_open` per host, and writes results to a `cpf-temporal-states` index.

5.3 Alert Fatigue Detection [5.x]

```
event.category:"alert" AND
  data_source:"crowdstrike"
```

Listing 6: KQL: Alert fatigue filter

Use a **Transform job** to compute the investigation rate per analyst per week. The Transform reads events matching the filter above, groups by `user.name` and a weekly time bucket, and computes `invest_rate = investigated / total`. It writes the result to a `cpf-fatigue-states` index. The CPF engine reads this index to detect decay patterns over time.

5.4 Convergence Index

In Elastic, the Convergence Index is best computed via a **Transform job** that reads from a dedicated `cpf-indicator-states` index (written to by each indicator's detection rule) and produces a single document per evaluation period containing the CI value and state.

6 SIEM Integration: Microsoft Sentinel

Sentinel uses Kusto Query Language (KQL) and integrates natively with Microsoft's ecosystem. CrowdStrike, Qualys, Tenable, and Rapid7 all have Sentinel connectors available through the Azure Marketplace or the Sentinel connectors gallery.

6.1 Temporal Vulnerability Detection [2.x]

```
CommonSecurityLog
| where DeviceVendor == "Qualys"
    or DeviceVendor == "Tenable"
| where DeviceEventClassID contains "vuln"
| extend DaysOpen =
    datetime_diff("day", TimeGenerated,
      todatetime(DeviceCustomString1))
| where DaysOpen > 90
| summarize
    OpenCount = count(),
    AvgDays = avg(DaysOpen)
    by DestinationHostName,
       DeviceCustomString2 // CVE
| extend CPF_Signal =
    iff(AvgDays > 180, "RED",
    iff(AvgDays > 90, "YELLOW", "GREEN"))
```

Listing 7: KQL (Sentinel): Temporal vulnerability

Note: `DeviceCustomString1--6` are mapped differently by each connector. Verify the exact mapping in your Sentinel workspace before deploying.

6.2 Alert Fatigue Detection [5.x]

```
SecurityAlert
| where ProductName == "CrowdStrike"
| extend WeekNumber =
    week_of_year(TimeGenerated)
| summarize
    TotalAlerts = count(),
    Investigated = countif(
      Status == "Investigated")
    by WeekNumber, AssignedTo
| extend InvestRate =
    Investigated * 1.0 / TotalAlerts
| extend CPF_Signal =
    iff(InvestRate < 0.3, "RED",
    iff(InvestRate < 0.6, "YELLOW", "GREEN"))
```

Listing 8: KQL (Sentinel): Alert fatigue

6.3 Convergence Index

Sentinel supports **Scheduled Alert Rules** that run on a defined schedule and write results to a custom table. Create a table `CPF_ConvergenceIndex` and a scheduled rule that reads from `CPF_IndicatorStates` (populated by each indicator's alert rule) and computes the CI using the same weighted formula as the Splunk version above.

7 SIEM Integration: QRadar

QRadar uses Ariel Query Language (AQL) for data retrieval. The CPF engine runs as a combination of saved searches and custom correlation rules within QRadar.

7.1 Temporal Vulnerability Detection [2.x]

```
SELECT
  t.qradar_source_name as scanner,
  t.source_ip,
  t.vulnerability_name,
  DATEDIFF('day',
    t.first_detected,
    NOW()
  ) as days_open
FROM events t
WHERE t.event_category = 'Vulnerability'
  AND t.vulnerability_status = 'Active'
  AND DATEDIFF('day',
    t.first_detected,
    NOW()
  ) > 90
ORDER BY days_open DESC
```

Listing 9: AQL: Temporal vulnerability

In QRadar, wrap this in a **custom correlation rule** that fires when the count of vulnerabilities with `days_open > 90` for a given host exceeds a threshold. The correlation rule outputs a CPF offense with a custom category that maps to indicator [2.x].

7.2 Alert Fatigue Detection [5.x]

```
SELECT
  t.username as analyst,
  COUNT(*) as total_alerts,
  SUM(CASE
    WHEN t.alert_status = 'Investigated'
    THEN 1 ELSE 0
  END) as investigated,
  CAST(
    SUM(CASE
      WHEN t.alert_status = 'Investigated'
      THEN 1 ELSE 0
    END) AS FLOAT
  ) / COUNT(*) as invest_rate
FROM events t
WHERE t.source_name = 'CrowdStrike'
  AND t.event_category = 'Detection'
GROUP BY t.username
HAVING CAST(
    SUM(CASE
      WHEN t.alert_status = 'Investigated'
      THEN 1 ELSE 0
    END) AS FLOAT
  ) / COUNT(*) < 0.3
```

Listing 10: AQL: Alert investigation rate

7.3 Convergence Index

QRadar's correlation engine supports **grouped correlation rules** that aggregate across multiple offense types. Create a grouped rule that watches for offenses tagged with CPF indicator categories. When three or more distinct categories are active simultaneously for the same organization, the rule fires with a CPF Convergence Index offense.

8 Convergence Index Computation

Regardless of SIEM platform, the Convergence Index follows the same logic. Each indicator has a state (Green, Yellow, Red) as defined by the Implementation Companion [2]. The CI aggregates these states into a single score.

8.1 Formula

$$CI = \frac{\sum_{i=1}^{n} w_i \cdot s_i}{\sum_{i=1}^{n} w_i}$$

where s_i is the numeric state of indicator i (Green = 0, Yellow = 0.5, Red = 1.0), and w_i is the weight defined in the Companion's Bayesian network for that indicator.

8.2 State Thresholds

Table 3: CI state thresholds

CI Range	State	Action
0.0–0.2	Normal	Routine monitoring
0.2–0.4	Elevated	Increase alert attention
0.4–0.6	High	Analyst review required
0.6–1.0	Critical	Incident response posture

8.3 Update Frequency

The CI should be recomputed every time any individual indicator changes state. In practice, this means the CI query runs on the same schedule as the slowest indicator query. For most deployments, a 1-hour cycle is sufficient for the first 30 days. Once the deployment stabilizes, this can be adjusted based on the organization's incident rate.

9 Output: Alerts and Dashboard Feed

The CPF engine produces two outputs. The first is an alert when any indicator transitions to Yellow or Red, or when the Convergence Index crosses a threshold. The second is a continuous state feed that populates a dashboard.

9.1 Alert Structure

Every CPF alert contains the following fields, regardless of SIEM platform:

9.2 Dashboard Feed

The dashboard reads from the same indicator state index that the CI query uses. It displays: current CI value and state, the count of indicators in each state (Red/Yellow/Green) per cate-

Table 4: Standard CPF alert fields

Field	Content
`cpf.indicator`	Indicator code (e.g., [5.1])
`cpf.category`	Category name
`cpf.state`	GREEN / YELLOW / RED
`cpf.previous_state`	State before transition
`cpf.ci_value`	Current Convergence Index
`cpf.ci_state`	CI state (Normal/Elevated/High/Critical)
`cpf.timestamp`	Evaluation timestamp

gory, and the trend of the CI over the last 30 days. The Deployment-Validation Loop [4] defines the maturity levels that map to dashboard views—an organization at Level 2 (Monitoring) sees the raw state feed; an organization at Level 4 (Proactive) sees lead-time predictions alongside the current state.

10 Categories Requiring Special Treatment

Four indicator categories do not map cleanly to the external data sources covered in Section 3. They require either native SIEM data, retrospective analysis, or both. This section covers each one.

10.1 [4.x] Affective States

Affective vulnerability indicators—shame, guilt, fear, anxiety in security decision-making—cannot be measured directly. No tool reports that an analyst felt ashamed. But the behavioral consequences of affective states are observable in the SIEM.

The primary proxy is **post-incident access pattern deviation**. When a security event implicates a specific user (as operator, not as target), that user's access behavior in the hours that follow deviates from their baseline in predictable ways if an affective state is active. Guilt-driven behavior produces increased access to the relevant systems—checking, correcting, covering. Fear-driven behavior produces avoidance—delayed logins, reduced activity, or delegation of tasks.

The following SPL query illustrates the detection pattern. Equivalent logic applies in other SIEMs using their native user activity data.

```
index=security_events
  event_type="incident"
| eval incident_user = involved_user
| eval incident_time = _time
| join type=inner incident_user [
    search index=authentication
    | stats count as login_count,
        avg(session_duration) as avg_session
      by user, bin(_time, "1h") as hour
  ]
| eval hours_since =
    (now() - incident_time) / 3600
| where hours_since <= 12
| eval baseline_logins =
    avg(login_count) over [
      search index=authentication
      | stats avg(login_count) as bl
        by user
    ]
| eval deviation =
    (login_count - baseline_logins)
    / baseline_logins
| eval cpf_signal =
    if(abs(deviation) > 2.0, "RED",
    if(abs(deviation) > 1.0, "YELLOW", "GREEN"))
```

Listing 11: SPL: Post-incident affective behavior proxy

10.2 [6.x] Group Dynamics

Group dynamics indicators target collective vulnerability patterns: normalization of risk, shared blind spots, and what Bion termed basic assumptions operating at the team level. These are not individual behaviors—they are patterns visible only when multiple analysts' decisions are examined together.

The detection proxy is **team-level alert disposition convergence**. When a team of analysts independently classifies similar alerts as low-priority within the same time window, and this convergence rate exceeds the team's historical baseline, the pattern is consistent with group normalization.

```
index=soc_analyst_actions
  action_type="alert_disposition"
| eval team = lookup(analyst_team,
    analyst_name)
| eval week = strftime(_time, "%Y-%W")
| stats count as dispositions,
    count(eval(priority="low")) as low_count
    by team, week, alert_category
| eval low_rate = low_count / dispositions
| join type=left team [
    search index=soc_analyst_actions
    | eval team = lookup(analyst_team,
        analyst_name)
    | eval week = strftime(_time, "%Y-%W")
    | stats count as d,
        count(eval(priority="low")) as l
      by team, week, alert_category
    | eval rate = l / d
    | stats avg(rate) as baseline_rate
      by team, alert_category
  ]
| eval norm_score =
    (low_rate - baseline_rate) / baseline_rate
| eval cpf_signal =
    if(norm_score > 0.5, "RED",
    if(norm_score > 0.3, "YELLOW", "GREEN"))
```

Listing 12: SPL: Group normalization detection

Note: this query requires that analyst actions (alert dispositions with priority assignments) are logged in the SIEM. Most SOC platforms log this data natively. If your SOC does not log individual analyst disposition decisions, this indicator cannot be computed.

10.3 [8.x] Unconscious Process

This is the only category with no real-time detection capability. The Validation Roadmap [3] identifies [8.x] as the hardest category: unconscious processes—shadow projection, rationalization, attribution distortion—are, by definition, not observable while they are active. They become visible only retrospectively, when the consequences of the distorted cognition can be compared against what the record shows actually happened.

The CPF engine does not produce real-time signals for [8.x]. Instead, [8.x] is evaluated through a **post-mortem analysis protocol**. After each security incident, the incident report is reviewed for patterns consistent with unconscious process indicators: attribution of the incident to external causes when internal factors were present, rationalization of decisions that were demonstrably risky at the time they were made, or projection of responsibility onto systems or other teams when the evidence points elsewhere.

This analysis is manual. It requires access to the full incident timeline, the communications that preceded the incident, and the post-mortem documentation. It is performed by a reviewer who was not involved in the incident response.

The output of this analysis feeds back into the Convergence Index retrospectively—it adjusts the CI for the period in question based on evidence that was not available in real time. This retrospective adjustment is logged in the `cpf-indicator-states` index with a flag indicating it was generated post-mortem rather than in real time.

10.4 [9.x] AI-Specific Bias

As organizations deploy AI-assisted tools within their SOCs—automated alert triage, incident summarization, risk scoring—these tools become both an asset and a new attack surface. Category [9.x] targets the vulnerability patterns specific to AI systems: susceptibility to prompt manipulation, training data exploitation, and systematic bias in AI-assisted decisions.

The detection proxy is **AI-human decision divergence**. When an AI tool's classification systematically disagrees with the analyst's subsequent manual classification on the same alert, and the pattern of disagreement is non-random, the AI tool may be operating under a biased or manipulated input condition.

```
index=ai_triage_logs
  ai_tool="*"
| eval ai_priority = ai_classification
| join type=inner alert_id [
    search index=soc_analyst_actions
    | rename priority as analyst_priority
  ]
| eval divergent =
    if(ai_priority != analyst_priority, 1, 0)
| stats count as total,
    sum(divergent) as divergences
    by ai_tool, alert_category,
       bin(_time, "1d") as day
| eval div_rate = divergences / total
| eval baseline_div = 0.15
| eval cpf_signal =
    if(div_rate > 0.4, "RED",
    if(div_rate > 0.25, "YELLOW", "GREEN"))
```

Listing 13: SPL: AI-human decision divergence

Note: the baseline divergence rate (0.15) is illustrative. Each organization should establish its own baseline from the first 30 days of AI tool operation. A divergence rate that exceeds the baseline by more than 2 standard deviations triggers a [9.x] signal regardless of the absolute value.

11 Deployment Checklist

The following checklist covers the minimum steps to get the CPF engine operational in a production SOC. Each step assumes the SIEM and data sources are already in place.

11.1 Phase 1: Data Verification (Day 1)

Verify that the required fields from Section 3 are present in your SIEM. Run a simple count query against each data source to confirm ingestion is active. If any data source is missing, the indicators that depend on it (Table 2) will not produce signals—this is expected and does not block deployment of the remaining indicators.

11.2 Phase 2: Indicator Deployment (Days 2–5)

Deploy the indicator queries from Sections 4–7 (whichever section matches your SIEM). Start with the three indicators that produce signal fastest: [2.x] Temporal (requires only vulnerability scan history), [5.x] Cognitive Overload (requires only alert data), and [1.x] Authority (requires only endpoint process data). These three categories cover the majority of observable pre-incident signals.

11.3 Phase 3: Convergence Index (Day 6)

Deploy the CI computation query. At this point, the system is in the state the Deployment-Validation Loop [4] defines as Level 2: Monitoring. Metrics are accumulating.

11.4 Phase 4: Alert Tuning (Days 7–30)

The first 30 days will produce false positives as the system calibrates to the organization's baseline. Do not adjust thresholds during this period—the noise is itself data. After 30 days, review the alert volume and precision. If precision is below 0.2, review the indicator queries against the Companion's detection logic to ensure the field mappings are correct.

11.5 Phase 5: Dashboard (Day 7 onward)

The dashboard can be deployed in parallel with Phase 4. It reads from the indicator state index and requires no additional configuration beyond the queries already deployed.

References

[1] Canale, G. (2025). The Cybersecurity Psychology Framework: A Pre-Cognitive Vulnerability Assessment Model. *CPF Technical Report Series.*

[2] Canale, G. (2025). Operationalizing the Cybersecurity Psychology Framework: A Systematic Implementation Methodology. *CPF Technical Report Series.*

[3] Canale, G. (2026). Toward Empirical Validation of the Cybersecurity Psychology Framework: A Tiered Methodological Roadmap. *CPF Technical Report Series.*

[4] Canale, G. (2026). From Framework to Operations: The CPF Deployment-Validation Loop. *CPF Technical Report Series.*