Detection Engineering Program Proposal

Prepared for: Cybersecurity Leadership Team

Prepared by: Adam Ring

Date: July 2025

Version: 2

Table of Contents

Proposal: Establishing a Detection Engineering Program

Prepared by: Adam Ring
Date: 7/30/2025
Audience: Executive & Technical Leadership

Why We Need a Detection Engineering Program

Cyber threats are increasingly sophisticated, targeted, and persistent. Traditional alerting, compliance-driven monitoring, and ad hoc detection development are no longer sufficient.

Current Challenges:

- Inconsistent coverage of adversary behaviors
- High false positive rates reduce SOC efficiency
- Lack of structured process to evolve detections
- Reactive rather than proactive posture

Business Risk:

- Extended dwell time of attackers
- Missed compliance requirements
- Undetected lateral movement and privilege abuse

Solution: A formal, repeatable Detection Engineering program.

Goals of the Program

- Threat-informed detection aligned with MITRE ATT&CK
- Continuous improvement through feedback loops
- Reduced mean time to detect (MTTD)
- Scalable and auditable detection lifecycle
- Central integration of red team, IR, CTI, and SOC inputs

Key Detection Engineering Frameworks

Strategic Frameworks

1. **MITRE ATT&CK** – Adversary TTPs and detection alignment
2. **MITRE D3FEND** – Countermeasure mapping to ATT&CK
3. **MITRE Engage** – Adversary engagement planning
4. **MITRE Insider Threat Framework** – Enhancing insider threat capability
5. **Detection Maturity Matrix (DMM)** – Detection engineering progression
6. **Threat Detection Maturity Framework** – Evaluate detection maturity
7. **ADS Framework** – Standardized alert and detection documentation
8. **Insider Threat Program Maturity Framework** – Best practices for internal risk

Industry Standards

1. **NIST SP 800-92** – Security log management guidance
2. **NIST SP 800-171/172** – Protection of Controlled Unclassified Information (CUI)
3. **NIST SP 800-61** – Incident handling and response
4. **NIST SP 800-86** – Forensic techniques in incident response
5. **CIS Benchmarks** – Secure configurations and system hardening

Detection Engineering-Specific Tools

1. **DeTT&CT** – Scoring log source and detection coverage quality
2. **Detection-as-Code** – Version-controlled, YAML-based rule development
3. **MaGMa Framework** – Use case management for security monitoring
4. **OTHF** – Open Threat Hunting Framework
5. **TaHiTI** – Threat hunting methodology
6. **RE&CT Framework** – Categorized incident response techniques

---

Detection Engineering Lifecycle

1. Intake

1. Detection requests from IR, CTI, SOC, red team, audits
2. Prioritize based on risk, coverage gaps, and business impact

2. Research

1. Understand relevant TTPs, behaviors, and threat actors
2. Validate data sources for coverage (via DeTT&CT, log analysis)

3. Build

1. Translate hypotheses into KQL or SIGMA/YAML rules
2. Apply ATT&CK mappings, metadata, alert logic, suppression filters

4. Test

1. Simulate threats using tools like Atomic Red Team or CALDERA
2. Validate true/false positive behavior with test data

5. Deploy

1. Code reviewed and pushed to main via GitHub CI/CD
2. Detection deployed into Sentinel, monitored with version control

6. Monitor

1. Evaluate alert quality and volume
2. Use analyst feedback and telemetry to refine logic

7. Maintain

1. Track rule decay, threat landscape shifts, and stale logic
2. Retire, improve, or replace based on feedback, hunts, and KPIs
3. Incident analysis and investigation
4. Response execution (containment, eradication, remediation)
5. Use of forensic tools and RE&CT for process adherence
6. Integration with CSIRT and lessons learned feedback loop

Operational Model Options

Option A: Centralized Detection Engineering Team

1. Dedicated engineers build and maintain detections
2. CI/CD, quality control, versioned pipeline

Option B: Hybrid Intake Model (Recommended)

1. SOC, IR, CTI, and Red Team submit detection requests
2. DE team triages, builds, and maintains quality

Option C: Federated Contributions with Governance

1. All teams contribute detection logic
2. DE sets standards and approves deployments

Metrics and KPIs

1. Time to detection (TTD) and alert-to-incident rate
2. MITRE ATT&CK tactic/technique coverage
3. False positive/false negative ratios
4. Number of rules tied to incidents or red team validation
5. Coverage of high-priority threat scenarios

Progress to Date: Engineering Milestones Completed

Proposed Implementation Timeline

Phase	Timeline	Milestone
Planning	Month 1	Team formation, metrics, framework adoption
Foundation	Months 2–3	CI/CD setup, repo creation, rule templates
Detection Build	Months 4–6	25+ rules from red team/IR threats
Evaluation	Month 7+	Metric review, purple teaming, roadmap updates

---

Next Steps: Detection Engineering Roadmap

The following key initiatives remain to fully operationalize and mature the Detection Engineering program:

Final Recommendation

1. Establish a Detection Engineering program under the Security org
2. Use MITRE ATT&CK, D3FEND, DMM, and NIST frameworks
3. Employ a hybrid detection intake model
4. Integrate detection-as-code, CI/CD, and operational metrics

Next Steps:

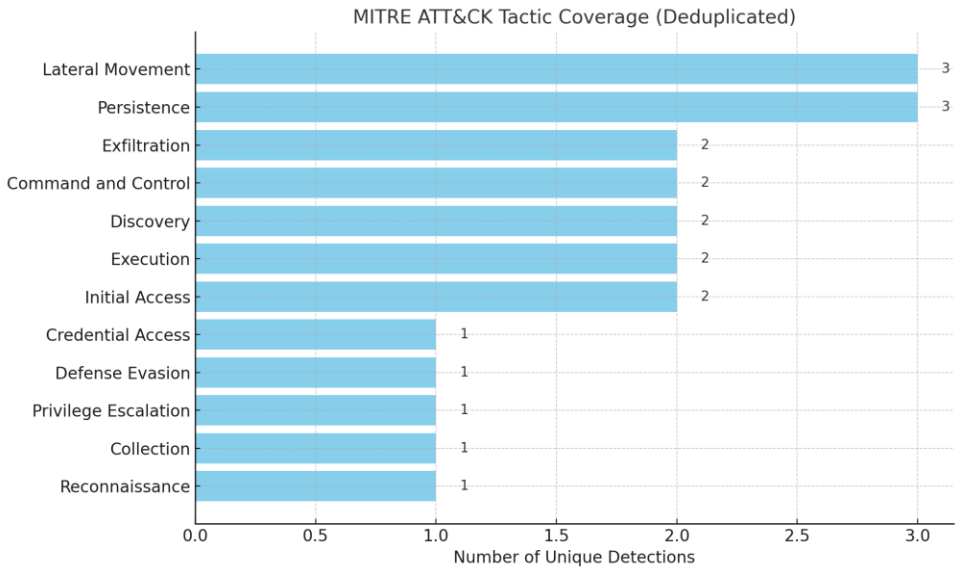
1. Executive sponsorship
2. Assign DE lead and engineering resources
3. Begin detection onboarding and tuning
4. Report on progress quarterly using maturity frameworks

Appendix: Supporting Frameworks & References

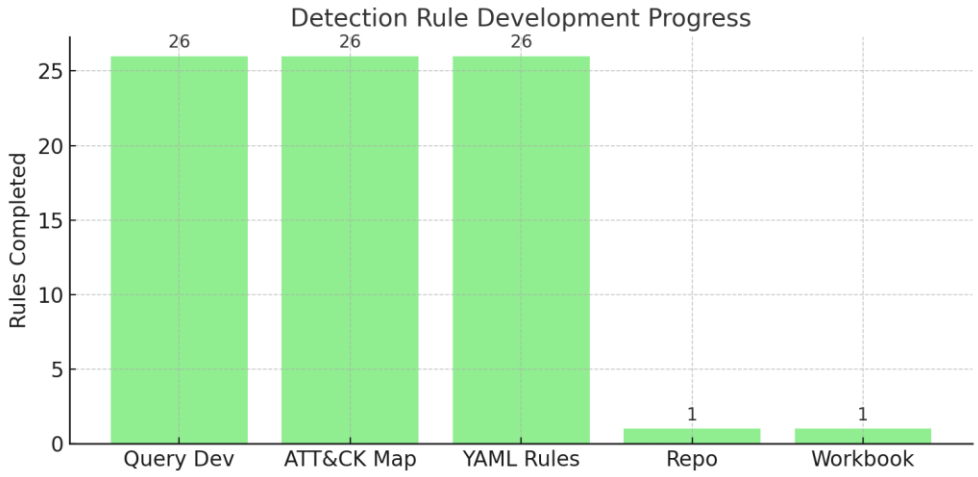
1. Snowflake – Detection DevOps best practices
2. NIST 800-92, 800-61, 800-171/172, 800-86
3. MITRE ATT&CK, D3FEND, Engage
4. DeTT&CT, OTHF, TaHiTI, MaGMa, RE&CT

Visual Summary of Detection Engineering Program

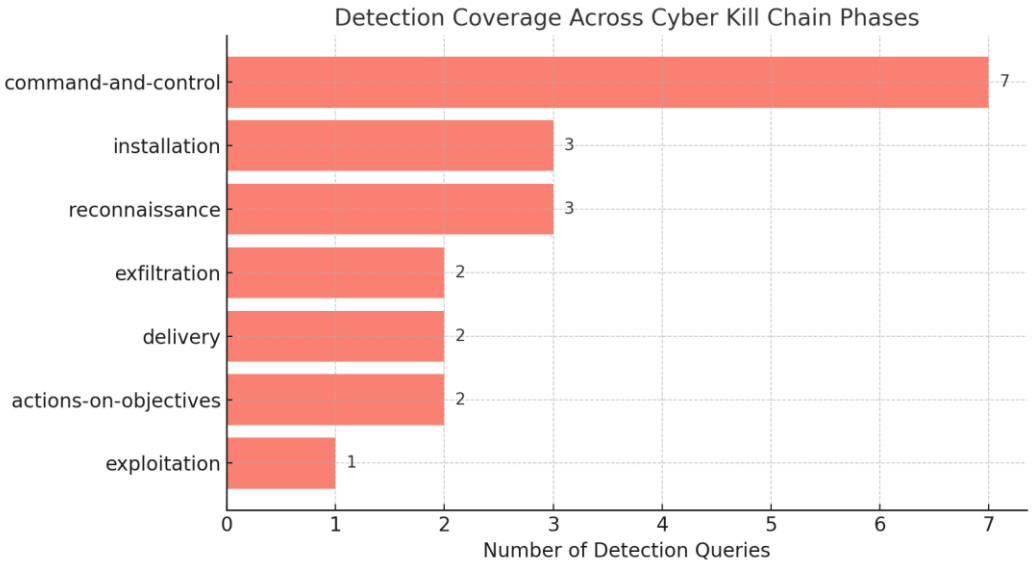
Detection Engineering Lifecycle



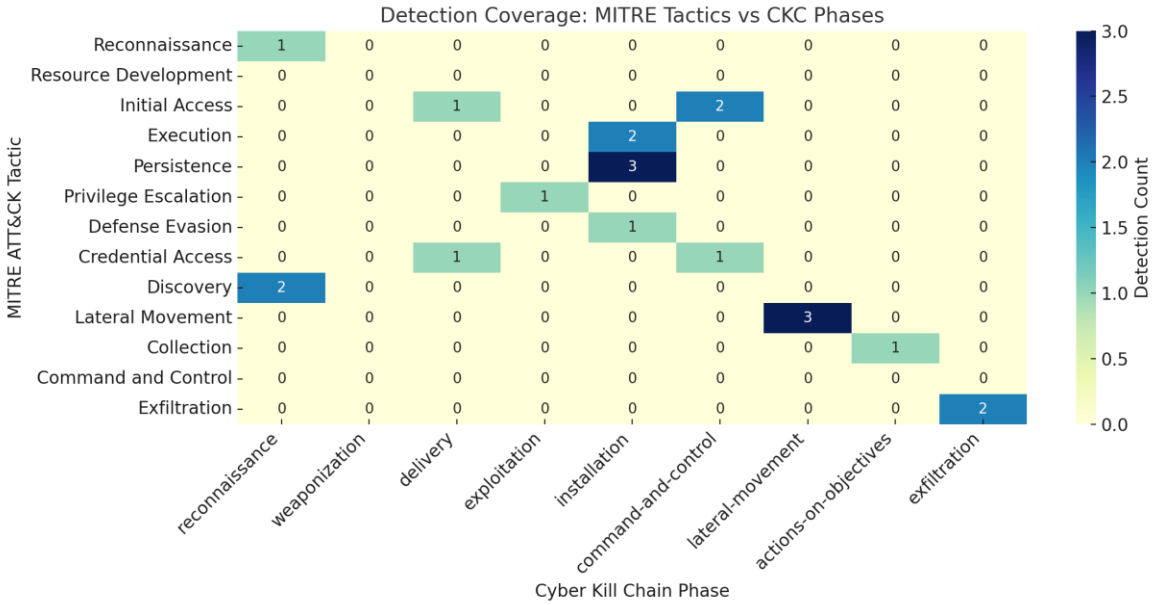
Detection Rule Development Timeline



Cyber Kill Chain Detection Coverage



Detection Coverage: MITRE Tactics vs CKC Phases



Progress to Date: Engineering Milestones Completed

As of July 30, 2025, the following core elements of the Detection Engineering program have been successfully completed:

Phase	Description	Status
1. Define Mission & Scope	Focused on MITRE ATT&CK and Cyber Kill Chain coverage	✅ Completed

	within Microsoft Sentinel and Defender ecosystem	
2. Data Ingestion	Logs from Microsoft Defender, Azure AD, Office 365, and Device Telemetry are integrated and verified	✅ Completed
3. Detection Query Development	Authored 31 KQL queries covering real-world TTPs across execution, credential access, defense evasion, and C2	✅ Completed
4. ATT&CK Mapping	Each query mapped to corresponding MITRE ATT&CK tactics and techniques, visualized in a Sentinel workbook	✅ Completed
5. Rule Conversion	All queries converted into YAML-based, version-controlled detection rules with ATT&CK metadata	✅ Completed
6. GitHub Repo Creation	Detection-as-code repository with organized folder structure, ready for CI/CD integration	✅ Completed
7. Workbook for Coverage	MITRE ATT&CK tactic/technique coverage workbook generated for tracking detection maturity	✅ Completed

Detection Engineering Lifecycle Status Overview

As of July 30, 2025, the following core elements of the Detection Engineering program have been successfully completed. This table outlines the complete lifecycle of the detection engineering program, indicating which phases are completed and which are in progress or pending:

Phase	Description	Status
1. Define Mission & Scope	Focused on MITRE ATT&CK and Cyber Kill Chain coverage	✅ Completed

	within Microsoft Sentinel and Defender ecosystem	
2. Data Ingestion	Logs from Microsoft Defender, Azure AD, Office 365, and Device Telemetry are integrated and verified	✅ Completed
3. Detection Query Development	Authored 31 KQL queries covering real-world TTPs across execution, credential access, defense evasion, and C2	✅ Completed
4. ATT&CK Mapping	Each query mapped to corresponding MITRE ATT&CK tactics and techniques, visualized in a Sentinel workbook	✅ Completed
5. Rule Conversion	All queries converted into YAML-based, version-controlled detection rules with ATT&CK metadata	✅ Completed
6. GitHub Repo Creation	Detection-as-code repository with organized folder structure, ready for CI/CD integration	✅ Completed
7. Workbook for Coverage	MITRE ATT&CK tactic/technique coverage workbook generated for tracking detection maturity	✅ Completed
8. CI/CD Deployment Pipeline	Build a GitHub Actions (or Azure DevOps) workflow to validate YAML syntax and push detection rules to Microsoft Sentinel	☐ In Progress
9. Alert Triage & Playbooks	Integrate detection rules with Sentinel playbooks to automate triage, enrich alerts, and initiate response actions	☐ Not Started

10. Validation & Simulation Testing	Use Atomic Red Team, Prelude Operator, or custom scripts to simulate adversary behavior and validate rule effectiveness	☐ Not Started
11. Tuning & False Positive Reduction	Refine logic using watchlists, suppression filters, and scoped exclusions based on telemetry and analyst feedback	☐ Not Started
12. Reporting & Dashboarding	Build dashboards to track alert volume, rule performance, and MITRE/Kill Chain coverage trends	☐ Not Started
13. Purple Teaming & Feedback Loops	Establish regular feedback between Red Team, CTI, IR, and Detection Engineering	☐ Not Started
14. Documentation & Knowledge Transfer	Develop rule documentation templates, tuning guides, and onboarding material	☐ Not Started

Appendix B: Palantir Detection-as-Code (DaC) Framework Recommendation

Recommendation: Adopt Palantir's Detection-as-Code Framework

Executive Summary

To enhance the maturity, agility, and consistency of our Security Operations Center (SOC), we recommend implementing a Detection-as-Code (DaC) approach using Palantir's open-source DaC framework. This strategy aligns with modern detection engineering best practices and will support scalable, auditable, and collaborative detection development, particularly as we expand our use of Microsoft Sentinel, Databricks, and hybrid cloud infrastructure.

Why Palantir Detection-as-Code

Palantir's DaC framework is a vendor-agnostic, open-source solution that helps security teams implement software engineering principles into detection engineering. It offers:

Version-Controlled YAML Detection Rules

Write detection logic in human-readable YAML.

Enables Git-based change tracking, code review, and rollback.

CI/CD Pipeline Integration

Automate validation, testing, and deployment of detection rules.

Enforces quality control and reduces configuration drift.

Modular, Reusable Components

Use macros, tags, and shared logic blocks.

Promotes consistent rule development across threat categories.

Cross-Platform Compatibility

Translate rules to KQL (Microsoft Sentinel), SQL (Databricks), SPL (Splunk), etc.

Allows for unified detection strategy across different platforms.

Improved Collaboration and Transparency

Encourages cross-functional contributions with pull requests, peer review, and Git issue tracking.

Enables structured handoffs between detection engineering, threat hunting, and incident response.

Palantir Detection Lifecycle

Palantir defines a structured lifecycle that guides the end-to-end process of detection engineering. This ensures detections are reliable, testable, and sustainable.

1. Ideation

Identify detection needs based on threat intel, incidents, purple teaming, or MITRE ATT&CK gaps.

Define hypotheses and threat behaviors.

2. Development

Write detection logic in YAML format using standard templates and macros.

Tag rules with MITRE ATT&CK techniques, severity levels, and relevant metadata.

3. Testing

Validate syntactic correctness (detection_rules validate).

Write rule-specific unit tests using sample logs or synthetic data.

Use adversary emulation (e.g., Atomic Red Team) for behavior verification.

4. Deployment

Automate deployment via GitHub Actions or Azure DevOps.

Translate and push queries into Sentinel or other SIEM platforms via APIs or Terraform.

5. Monitoring & Feedback

Monitor rule performance (true/false positive rates).

Feed detection feedback into backlog for tuning or retirement.

6. Maintenance

Review and refactor rules based on telemetry changes or evolving TTPs.

Ensure detection coverage is preserved as infrastructure evolves.

This lifecycle is repeatable and auditable, allowing security teams to treat detections with the same rigor as production code.

Implementation Plan

Step 1: Repository Setup

Create a GitHub or Azure DevOps repo (e.g., detection-engineering).

Structure folders for /rules, /tests, /macros, /deployment, /configs.

Step 2: Rule Development

Convert existing Sentinel KQL rules into YAML.

Enrich with MITRE tags, owner fields, and macro logic for reusability.

Step 3: CI/CD Integration

Set up a pipeline for:

Syntax validation

Unit test execution

Rule translation and deployment to Sentinel/Databricks

Step 4: Lifecycle Enforcement

Integrate the detection lifecycle into SOC SOPs.

Require test coverage, review approvals, and deployment logs before promotion.

Step 5: Dashboard Integration

Use MITRE tags to visualize coverage in Microsoft Sentinel workbooks.

Automate reporting on rules by ATT&CK technique, owner, last update, and coverage gaps.

Conclusion

Adopting Palantir's Detection-as-Code framework and lifecycle will allow us to transform detection engineering from ad-hoc query writing into a scalable, testable, and collaborative discipline. This approach enhances detection fidelity, speeds up deployment, and supports a resilient and future-proof SOC.

Appendix C: Detection Engineering Behavior Maturity Model (DEBMM) Assessment

DEBMM Overview

• Elastic's DEBMM defines five maturity levels—Foundation (Tier 0), Basic (Tier 1), Intermediate, Advanced, and Expert (Tier 4). It emphasizes structured development and continuous improvement of detection engineering behaviors over time.

• (Reference: https://www.elastic.co/security-labs/elastic-releases-debmm)

Current DEBMM Tier Alignment

• Tier 1 – Basic:

• Version-controlled and documented rule library in GitHub

• ~15–30 YAML rules with MITRE ATT&CK and Cyber Kill Chain tags

• Minimal stakeholder review, structured metadata present

• Tier 2 – Intermediate (In Progress):

• Heatmap coverage mapping and workbook correlation

• Detection suppression and watchlist filtering

• Consistent lifecycle tagging and auditability in YAML

• Tier 3 – Advanced (Partial):

• Behavioral simulation in lab environments

• Threat-informed rule enhancements in development

• Tier 4 – Expert (Planned):

• No current automation or SOAR integration

• No auto-deploy or feedback-driven suppression pipelines

DEBMM Tier Summary

- Tier 0 – Foundation: Ad hoc rules, no structure — ✅ Surpassed
- Tier 1 – Basic: Consistent naming, small ruleset, version control — ✅ Completed
- Tier 2 – Intermediate: Review process, tuning, data coverage — ✅ In Progress
- Tier 3 – Advanced: Threat emulation, feedback loops — ⚠️ Partial
- Tier 4 – Expert: Full automation, SOAR integration — ☐ Planned

Recommendations to Progress

- 1. Establish routine false-positive reviews and tuning.
- 2. Correlate hunt results with detection gaps.
- 3. Begin integration with automated playbooks and response tools.
- 4. Tag detection rules with response strategies and confidence levels.
- 5. Consider use of AI-assisted detection refinement and tuning.

Appendix D: Priority Data Sources for Detection Engineering

This appendix identifies and prioritizes core data sources required to support high-fidelity detections, threat hunting, and incident response across the environment. Focus initial onboarding, enrichment, and quality assurance on the sources below.

- Assets:
  - User endpoints (laptops, desktops, mobile devices)
  - Servers:
    - Are any of these servers public-facing?
    - What type of servers are they? Domain controllers, web servers, file servers, etc.
- Software:
  - Operating systems running on the various assets
  - Software/applications used throughout the organization
- Network architecture diagram
- Cloud assets:
  - SaaS applications
- Security solutions:
  - SIEM
  - Endpoint protection (AV/EDR)
  - Network protection (IPS/firewall/WAF)