Detection Engineering

Table of Content

1. Introduction

In today's rapidly evolving threat landscape, Detection Engineering has emerged as a critical discipline in cybersecurity, enabling organizations to identify and respond to malicious activities.

Detection Engineering is not just about writing detection rules, it is a structured approach that involves understanding adversary behavior, designing detection logic, continuously refining detection coverage, and integrating detections into security operations workflows. By leveraging modern security tools such as SIEM (Security Information and Event Management), EDR/XDR (Endpoint/Extended Detection and Response) and threat intelligence platforms, log analytics systems, organizations can systematically craft, test, and optimize detections to uncover both known and emerging threats.

Detection Engineering Benefits:

Improved Threat Detection: Detection Engineering focuses on behavior-based detection, allowing organizations to better identify threats.

Leverages the MITRE ATT&CK Framework: By mapping detections to MITRE ATT&CK techniques, security teams ensure comprehensive coverage of real-world attack behaviors.

Reduced False Positives & Alert Fatigue: Detection Engineering helps build detections that are precise, contextualized, and high-fidelity, reducing unnecessary alerts and improving security operations efficiency.

Empowers Threat Hunting: Detection Engineering enhances threat hunting by providing high-fidelity use cases for proactive threat identification.

Enhances Log Visibility: Effective detection relies on clear visibility into logs, telemetry, and data sources. Detection Engineering ensures that organizations collect and analyze the right logs without overwhelming their security tools with unnecessary data.

Forces Attackers to Work Harder: By detecting and responding to Tactics, Techniques, and Procedures (TTPs) rather than Indicators of Compromise (IOCs), attackers must significantly alter their methods to avoid detection.

2. Detection Lifecycle

The most effective approach for creating and maintaining detections is by adhering to a structured, methodical process.

The Detection Lifecycle is a structured approach to developing, testing, deploying, and refining security detections. It ensures that security teams continuously improve their detection capabilities by iterating through a cycle of threat intelligence analysis, rule creation, validation, deployment, monitoring, and tuning.

The Stages of the Detection Lifecycle:

- Discovery
- Research
- Development
- Testing
- Deployment
- Continuous Tuning & Improvement

2.1. Discovery

The initial step in the lifecycle is gathering detection requirements. This phase can be initiated from multiple sources. Common sources include SOC requests, red team assessments, and threat intelligence reports, each contributing valuable insights into potential threats that require monitoring.

2.2. Research

This phase is essential for defining the detection logic to be implemented and identifying the necessary telemetry (Event Logs, Mirrored network traffic, etc.). If multiple detection requirements exist, this stage also involves prioritizing them to ensure the most critical threats are addressed first.

2.3. Development

Building upon the insights gained from the Research phase, the objective here is to implement the previously identified detection logic. This translates research findings into practical threat detection rules, moving from theoretical understanding to operational capability.

2.4. Testing

This process involves testing with two key data types: known good data, which helps prevent false positives by ensuring benign activities do not generate alerts, and known bad data, which confirms that the detection effectively identifies malicious behavior.
While both data types have their complexities, a general best practice is to ensure that known good data is sourced from an environment that mirrors the target environment as closely as it can, while known bad data is typically obtained through adversary emulation exercises.

This phase may also incorporate unit testing within CI (Continuous Integration) pipelines to verify that specific requirements, such as the presence of required fields and the correctness of their values, are met.

2.5. Deployment

Deploying the finalized detection rule into a production environment. This process is most effective when implemented through a Detection-as-Code (DaC) pipeline leveraging CD (Continuous Deployment) automation to ensure seamless and controlled deployment.

2.6. Continuous Tuning & Improvement

It is important to ensure detections remain effective, accurate, and relevant over time. Threat actors continuously evolve their TTPs (Tactics, Techniques, and Procedures), and if detections are not regularly updated, they risk becoming outdated, leading to false negatives (missed threats). Changes in the monitored environment can over time result in excessive false positives which also necessitate detection rule tuning to prevent alert fatigue.

3. Prioritizing Detection Efforts

Prioritization plays a crucial role in Detection Engineering by helping us address two key questions: What should be detected, and how?

3.1. Pyramid of Pain

Created by David Bianco the Pyramid of Pain visually describes the difficulty adversaries face when defenders focus detection efforts on different types of threat indicators. The pyramid categorizes indicators from the least to most difficult for attackers to alter:

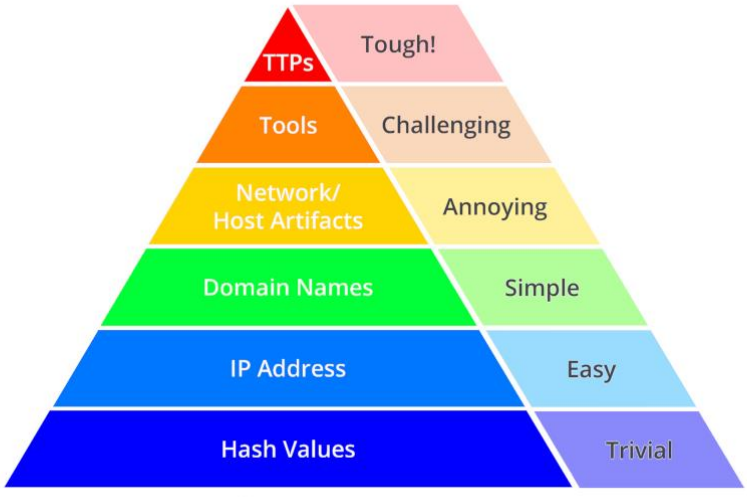

Figure 1: The Pyramid of Pain

By leveraging the Pyramid of Pain, organizations can shift detection efforts towards higher-value indicators—specifically, techniques and behaviors at the top of the pyramid that require attackers to expend considerable effort to change.

Conversely, focusing on easily changed indicators (such as hashes or IP addresses) results in limited defensive value, as adversaries can quickly evade detection by making minor modifications.

The higher you move on the pyramid:

- The more strategic and effective your defenses become.
- The greater the operational burden placed on attackers.
- The longer lasting your defensive measures become, resulting in greater resilience.

3.2. MITRE ATT&CK Heatmaps: Threat-Informed Prioritization

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework is a globally recognized knowledge base that documents real-world adversary tactics, techniques, and procedures (TTPs) used during cyberattacks. Developed and maintained by MITRE, ATT&CK provides a structured way for security teams to understand, analyze, and defend against cyber threats. Tactics represent the high-level objectives that adversaries aim to achieve during an attack. Techniques describe specific methods adversaries use to achieve their objectives (tactics). Procedures describe how specific threat actors or malware use these techniques in real-world attacks.

A MITRE ATT&CK Heatmap is a visual representation of an organization's detection coverage across different ATT&CK techniques. Heatmaps help security teams prioritize detection engineering efforts by highlighting:

- Which techniques are well-detected?
- Where are detection gaps?
- What techniques are commonly used by adversaries targeting the organization?

An example of a heatmap is the TI-Based Retrospective TTP Heatmap available on github.

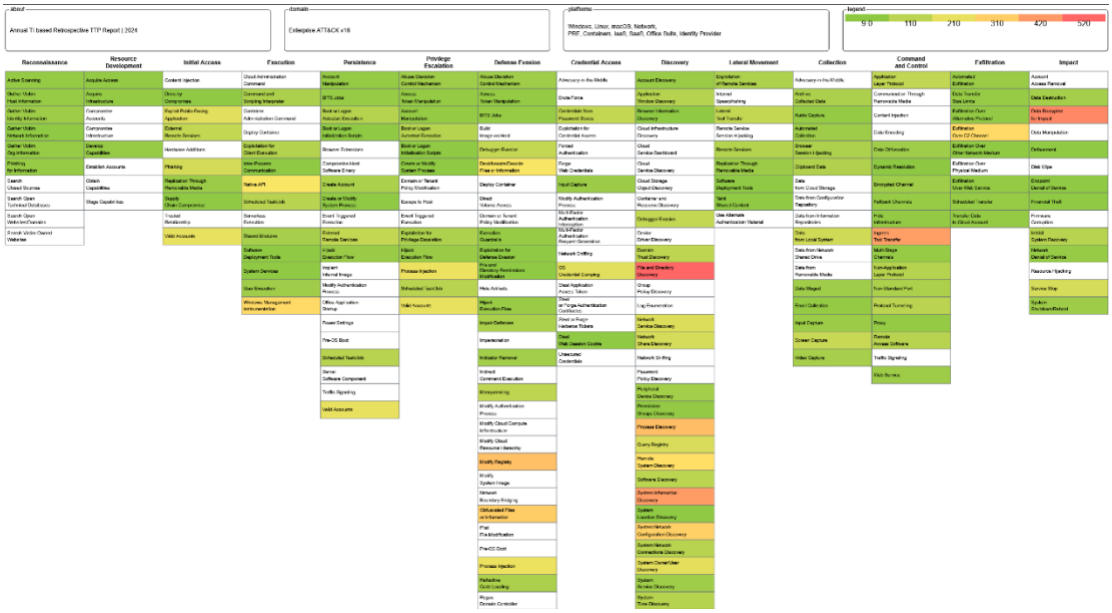

Figure 2: The Annual_TI_based_Retrospective_TTP_report_heatmap

4. Alert Classification

Alert classification is the process of categorizing the security events that have (or haven not) been generated by detection rules.

Events generally fall into four categories:

- True Positives (TP)
- False Positives (FP)
- True Negatives (TN)
- False Negatives (FN)

True Positives occur when a detection correctly identifies malicious activity, allowing security teams to respond appropriately.

False Positives, on the other hand, are benign activities mistakenly flagged as threats, leading to unnecessary investigations and alert fatigue. This type is typically the most common. False positives are not necessarily a problem, as long as they are properly managed and kept under control.

True Negatives represent legitimate activity that is correctly ignored, ensuring the system does not generate unnecessary alerts.

False Negatives are the most dangerous, as they indicate missed threats—malicious actions that bypass detection, leaving the organization vulnerable.

Striking the right balance between reducing false positives while minimizing false negatives is a key challenge in Detection Engineering, requiring continuous rule maintenance.

5. Detection Engineering Principles

The objective of Detection Engineering is to develop efficient, scalable, and high-fidelity detections.

To achieve this, it is essential to follow key principles:

- Prioritize inclusion-by-exception over exclusion-by-exception to avoid an endless cycle of reactive adjustments.
- Utilize correlation strategically but avoid overly broad rules that may introduce brittleness and enable easy evasion.
- Focus on detecting behaviors rather than IOCs, unless IOC-based detection is the only viable approach.
- Ensure simplicity in rule design, as overly complex rules lead to even more difficult-to-interpret alerts.
- Reducing false positives can sometimes take priority over eliminating false negatives, as excessive alerts can lead to analyst fatigue and decreased operational efficiency.
- Detection logic should be formatted to clearly reflect logical precedence and grouping for better readability and maintainability.

6. Detection Metrics

Detection metrics serve as a quantifiable approach to evaluating the performance of security detections, helping organizations assess the efficiency, accuracy, and overall impact of their detection strategies.

The most critical detection metrics include:

Alert count: This metric represents the number of alerts generated by a detection that requires analyst review. A high alert count may indicate a noisy or overly broad detection rule, leading to increased false positives and alert fatigue.

Average Time Spent per Detection: Measures the average time an analyst spends investigating alerts triggered by a specific detection. High time per detection suggests insufficient context, missing evidence, or poor alert documentation, forcing analysts to spend more effort investigating. Low time per detection could indicate that analysts are not thoroughly reviewing alerts, possibly leading to oversight.

Detection Coverage: Evaluates how well an organization's detections map to real-world attack techniques and adversary behaviors. Often visualized using MITRE ATT&CK heatmaps, this metric helps identify gaps in detection capabilities, ensuring the organization monitors key attack vectors. A low detection coverage score means that certain MITRE ATT&CK techniques are not effectively detected, leaving blind spots in the security posture.

7. Detection Formats

Storing detection content effectively is crucial for ensuring scalability, automation, and maintainability in Detection Engineering.

7.1. Sigma

Detection logic should be stored in Sigma, an open-source, generic signature framework specifically designed for writing SIEM-agnostic detection rules in YAML.

It allows security teams to create structured detection rules that can be converted into query formats specific to different SIEMs and log analysis platforms (e.g., Splunk, Elastic, Sentinel, QRadar). Instead of writing separate queries for each tool, Sigma rules provide a standardized way to define detections, making them highly portable and reusable.

Public Sigma translators, such as sigconverter.io or uncoder.io, allow users to convert Sigma rules into platform-specific queries, enabling seamless integration across various security tools. By leveraging Sigma, organizations can improve detection consistency, enhance threat hunting, and simplify rule management across multiple security environments.

7.2. TOML

A highly effective approach to storing detection content is by using TOML files. TOML (Tom's Obvious, Minimal Language) is a lightweight, human-readable configuration format designed for clarity, simplicity, and ease of use. It is widely adopted for configuration files across various programming environments due to its structured yet straightforward syntax. Compared to JSON and YAML, TOML offers improved readability while maintaining strong data structure support. Additionally, TOML is highly script-friendly, making it an excellent choice for automating detection processing within CI/CD pipelines, ensuring seamless integration and validation of detection logic.

8. Adversary Emulation for Detection Validation

Adversary emulation is the process of simulating real-world cyber threats by replicating the tactics, techniques, and procedures (TTPs) of known threat actors. It is used to test an organization's detection capabilities, incident response readiness, and security controls.

Atomic Red Team is an open-source adversary emulation framework developed by Red Canary. It provides lightweight, easy-to-execute tests that simulate real attack techniques mapped to MITRE ATT&CK. These tests allow security teams to safely validate detections, tune alerts, and improve defensive strategies without the complexity of full-scale red teaming.

Atomic Red Team is highly extensible, allowing users to easily create and customize their own atomic tests to simulate specific attack techniques.

Atomic tests are stored in YAML format, which may not always be the most intuitive or user-friendly to work with. To address this, AtomicGen was developed, providing a GUI-based approach.
To run tests, an execution framework is also needed. The most commonly used tool for this purpose is Invoke-Atomic, a PowerShell-based framework designed specifically for executing Atomic Red Team tests.

9. Detection Management

Detection as Code (DaC) is a modern approach to developing, managing, and maintaining security detections using software engineering principles. It treats detection rules and logic as structured code, enabling security teams to apply version control, automation, testing, and continuous integration (CI/CD) practices to threat detection. By implementing DaC, organizations can improve detection accuracy, streamline workflows, and ensure consistent threat monitoring across multiple security tools.

9.1. Core Principles of Detection as Code

- Version Control & Collaboration: Uses Git or similar version control systems to track changes, allowing teams to collaborate on detection rules and maintain an audit trail of modifications.
- Automation & CI/CD: Enables automated testing, validation, and deployment of detections, ensuring they are effective before production deployment.

- Scalability & Consistency: Provides a standardized approach to managing detections across multiple security tools and environments.
- Testing & Validation: Ensures detection rules are accurate, high-fidelity, and optimized before deployment, reducing false positives and negatives.

9.2. Example Detection as Code pipeline

A representative example of a DaC pipeline is structured as follows. Detection rules are written in a structured format, such as TOML files, and stored in a version-controlled Git repository. Version control ensures that all changes are tracked, reviewed, and auditable, allowing security teams to collaborate effectively while maintaining a history of rule modifications.

Once a new detection rule is added or an existing rule is updated, the CI/CD pipeline automatically initiates a validation process. This process includes syntax checks and rule data validation to ensure the rule functions correctly.

If the validation phase is successful, following a peer-review the pipeline automatically deploys the validated detection rules to production environments, such as SIEMs, EDRs, or XDR platforms. This ensures that new or updated detections are seamlessly integrated into the organization's security monitoring infrastructure without requiring manual intervention.