SIGMA Under the Microscope: A Longitudinal Analysis of Detection Rule Evolution

Problem Statement

- Community-driven SIGMA rules form one of the most widely used vendor-neutral detection content sources for SIEMs, EDRs, and XDR pipelines.
- Despite their importance, there is **no systematic understanding** of:
    - How SIGMA rules evolve over time.
    - How contributors maintain or modify rules after creation.
    - How the rules respond to emerging threats, CVEs, and ATT&CK updates.
    - What factors correlate with stable, high-quality, or correct rules.

Knowledge Gaps

Our study will address the following open gaps in the literature and practice:

- No longitudinal analysis exists of SIGMA rules as software artifacts that evolve, change, and drift.
- No empirical taxonomy of SIGMA rule modifications (logic updates, metadata fixes, false-positive mitigation, ATT&CK corrections).
- No understanding of rule lifecycle (initial creation, refinement period, stability).
- No measurement of community responsiveness to major vulnerabilities, new ATT&CK techniques, and threat reports.
- No quantitative assessment of coverage gaps across cloud, container, Linux, and emerging threat domains.
- No formal analysis of contributor behavior, expertise distribution, or how contributor experience affects rule quality.
- **No tool support exists for predicting unstable or error-prone rules or for scoring rule quality based on historical patterns. [This could be one of our main contributions]**

Concrete Steps for SIGMA Longitudinal Study

Part A: Data Collection

Step 1: Clone the SIGMA repository with full git history using `git clone https://github.com/SigmaHQ/sigma.git`.

Step 2: Extract commit-level metadata for all YAML rule files using `git log` with format strings to capture commit hash, author, email, date, and message for every commit touching `*.yml` files.

Step 3: Build a rule snapshot database by reconstructing each rule file's state at every commit where it changed, storing rule ID, timestamp, full YAML content, and diff from previous version using Python's `gitpython` or `pydriller` library.

Step 4: Parse all rule versions into structured format by extracting fields including title, id, status, logsource, detection logic, ATT&CK tags, references, author, and modification dates into SQLite or PostgreSQL.

Step 5: Compute diffs between consecutive versions of each rule and categorize what changed (detection logic, metadata, ATT&CK tags, logsource, false-positive notes).

Step 6: Collect external reference data including ATT&CK release dates from MITRE, CVE publication dates from NVD for referenced CVEs, and publication dates of major threat reports (Mandiant, CISA advisories).

Part B: Research Analyses

Analysis 1 (Rule Lifecycle): Measure time from rule creation to first modification, total number of modifications per rule, and time to stability (defined as no changes for 6+ months), then plot survival curves.

Analysis 2 (Modification Taxonomy): Manually label 200-300 rule changes into categories (logic fix, FP mitigation, metadata update, ATT&CK correction), then train a classifier or use heuristics to label remaining changes.

Analysis 3 (Threat Responsiveness): For major CVEs (Log4j, ProxyLogon, ProxyShell) and new ATT&CK techniques, measure the lag time from external event publication to first SIGMA rule addressing it.

Analysis 4 (Coverage Gaps): Count rules per logsource category (Windows, Linux, Cloud, Container) and map all rules to the ATT&CK matrix to identify sparse technique coverage.

Analysis 5 (Contributor Behavior): Identify top contributors, measure their experience (prior commits), and test whether contributor experience correlates with rule stability and fewer subsequent modifications.

Analysis 6 (Quality Prediction Model): Define "unstable" rules as those with many modifications or reverted changes, extract features (rule complexity, logsource type, contributor experience, initial status), and train a classifier to predict which new rules will require fixes.

Tools

Use Python with pandas, gitpython/pydriller, and scikit-learn for data processing and modeling. Use SQLite for storage and matplotlib/seaborn for visualization. Use the `mitreattack-python` library for ATT&CK mapping.