

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Credible Threat Detection? Measuring Contribution Dynamics and Quality Control in a Crowdsourced Threat Detection Ecosystem

Apurva Virkud, Gang Wang, and Adam Bates, *University of Illinois Urbana-Champaign*

https://www.usenix.org/conference/usenixsecurity26/presentation/virkud

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

Credible Threat Detection? Measuring Contribution Dynamics and Quality Control in a Crowdsourced Threat Detection Ecosystem

Apurva Virkud, Gang Wang, Adam Bates
University of Illinois Urbana-Champaign
{avirkud2, gangw, batesa}@illinois.edu

Abstract

Community exchange serves as a critical component of modern day security operations. Commercial security vendors often draw from crowdsourced intelligence and rules to power their proprietary systems. Additionally, Managed Security Service Providers (MSSPs) and Managed Detection and Response (MDR) solutions leverage both crowdsourced and commercial feeds to provide security services to organizations without their own security infrastructure. However, there is no systematic understanding of how these crowdsourced detection rules are developed or vetted. In this work, we explore the open-source and crowdsourced Sigma repository as a window into the development of threat detection rules. We find that the Sigma repository continuously iterates upon its rules, especially with tuning of false positives based on feedback from an active downstream userbase. Through an investigation of the quality control process, we observe evidence of informal rule evaluation, but also of inexperienced contributions from the broader community. Finally, we perform a comparative analysis of rules that have migrated between Sigma and other commercial Security Information and Event Management (SIEM) vendors (and vice versa) and observe divergences across platforms. Although crowdsourcing allows the community to iterate on commercial rules, we anecdotally observe that such contributions can introduce increased risk of rule evasion from the adversary. While Sigma is an active resource for vendors and the broader security community, it can benefit from more systematic and automated quality control procedures. As a first step, we introduce a tool to identify incoming duplicate rules to aid in the review process.

1 Introduction

Community exchange has always played an important role in threat detection. Dating back to hardcopy magazines circulating antivirus signatures in the 1980's [34], numerous platforms and marketplaces now exist to aggregate threat intelligence from various sources (e.g., [21,26,27,41,47,58,72]). It is sometimes the case that structured threat intelligence arrives as a fully-implemented detection analytic, e.g., a YARA rule [71] may be directly installable into an antivirus product. At other times, practitioners must review qualitative or semi-structured reports to implement a concrete analytic. This is especially the case for rule-based Endpoint Detection & Response (EDR) and Security Indicator & Event Management (SIEM) products. Instead of a byte sequence, detection analytics in these products encode behavioral logic that is matched against product-specific event telemetry streams, making the general distribution of fully-implemented analytics more difficult.

More recently, a community standardization effort for EDR and SIEM has emerged – Sigma [62], an open source repository for *deployable* threat detection rules. Sigma's initial purpose was to provide a standardized format for SIEM rules, analogous to YARA [71] for malware and STIX [54] for threat intelligence. Sigma specifies generic event fields for detection rule logic that can then be ported to product-specific backend formats like Elastic, Splunk, Crowdstrike, and Rapid7. Through succeeding in its standardization goal, Sigma has also become a hub for the detection rules. Sigma has been active for eight years, and today is growing rapidly in popularity. At present, the repository has centralized over 3,000 detection rules from hundreds of contributors.

Sigma's openness stands in stark contrast to many commercial EDR vendors, whose proprietary rule sets are core to the value of the product. Most EDR products only expose their rulesets to customers (e.g., [19]), or even conceal the existence of a rule until the moment a detection occurs (e.g., [59]). Threat detection rules can also be implemented as automated queries in SIEM products, a market in which vendors have been more open. Platforms like Splunk [64] and Elastic [25] have published open-source versions of their rulesets, but the process through which these rules were actually developed remains opaque. Thus, Sigma's fully-open development cycle grants a unique view into the practice of threat detection engineering. Simultaneously, the open and crowdsourced nature of Sigma's ruleset raises questions as to its maturity, compre-

```yaml
title: Potential CVE-2021-26857 Exploitation Attempt
id: cd479ccc-d8f0-4c66-ba7d-e06286f3f887
status: stable
description: Detects possible successful exploitation
   for vulnerability described in CVE-2021-26857 by
   looking for | abnormal subprocesses spawning by
   Exchange Server's Unified Messaging service
references:
  - https://www.microsoft.com/security/blog/2021/03/02/
   hafnium-targeting-exchange-servers/
author: Bhabesh Raj
date: 2021-03-03
modified: 2023-02-07
tags:
  - attack.t1203
  - attack.execution
  - cve.2021-26857
  - detection.emerging-threats
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    ParentImage|endswith: '\UMWorkerProcess.exe'
  filter:
    Image|endswith:
      - 'wermgr.exe'
      - 'WerFault.exe'
  condition: selection and not filter
falsepositives:
  - Unknown
level: high
```

Figure 1: An example threat detection rule from Sigma.

hensiveness, and reliability compared to commercial vendors.

In this work, we aim to investigate the development of crowdsourced threat detection engineering through the lens of the Sigma project and ruleset. *Through longitudinal analysis of Sigma's full commit history from 2017 to 2025,* we explore four main research questions:

RQ1 (§4): What are the dynamics of rule contribution? We analyze how the Sigma ruleset has evolved since its creation in late 2017. We find that rules are primarily added and modified, but *rarely deleted or deprecated*. Further, we find that the rules' actual detection logic continue to be iterated upon after their creation, primarily to address false positives in 61.6% of cases. Rule metadata, which also plays an important role in rule interpretation, is updated with greater frequency than the detections themselves. Rules' tags and status are the most frequently modified metadata fields, respectively reflecting the integration of external frameworks like MITRE ATT&CK and the year-long empirical testing process that rules experience. In comparison with commercial vendors, at a high level, both Elastic and Splunk have similar longitudinal trends where they primarily add and modify rules. Elastic and Splunk median response time to new CVEs is 4 and 13 days, respectively, compared to Sigma's 14 days.

RQ2 (§5): Who are the rule contributors and downstream users? We characterize the community surrounding the Sigma repository to understand the impact and crowdsourced nature of its detections. We analyze contributing authors, editors, and organizations, along with repository forks. We find that the maintainers are heavily involved in both authoring (55.6% of rules) and editing rules (72.7% of rules). Simultaneously, the impact of crowdsourcing on Sigma is also clear, including the presence of contributions from security vendors and the industry at large. We also find organizational accounts among the passive and diverging forks, suggesting that Sigma is a resource whose influence extends far beyond its active contributors.

RQ3 (§6): What quality control processes are in place? We analyze repository pull requests (PRs) to explore the review and quality control processes for rule contributions. We find that new pull requests have been moving towards more systematic review procedures within the last two years. Maintainers are able to respond to and resolve new pull requests within a day on average, with outliers attributable to inactivity from the PR author. While the review process primarily occurs between the author and maintainer, 11.7% of pull requests receive discussion from the broader community. We also observe evidence of rule evaluation, with 5.5% of PRs mentioning false positives in the comments. We conduct an additional analysis of the *rejected PRs* to understand why they are rejected. Again, we observe evidence of rule quality evaluation, although effective reviewing tends to rely on the small team of maintainers' institutional knowledge, with 12.5% PRs rejected due to the maintainers' manual identification of overlapping or duplicate rules.

RQ4 (§7): How does Sigma compare to major commercial systems? Finally, we aim to compare the Sigma ruleset to those of commercial open-source vendors, namely Elastic [25] and Splunk [64]. We identify that 71 Sigma rules have common ancestors or converge with Elastic (55) and Splunk (16) rules and perform a comparative analysis. We find that 64.8% rules do not have semantic differences between Sigma and the commercial vendor, while the remaining differences can be attributed primarily to false positive refinement. We investigate several case studies that demonstrate the strengths and weaknesses of crowdsourced threat detection.

Disclosure We disclosed our results to the Sigma maintenance team, who confirmed that Sigma is popular amongst contracted security providers (e.g., MSSPs). This complements our findings of traditional security vendors contributing to and ingesting Sigma rules. Further, it emphasizes the need for more systematic and robust *quality control* procedures, as these rules have the potential for widespread downstream impact across organizations.

Operationalizing Our Findings Overall, we find that Sigma's revew process can benefit from (semi-)automated procedures to reduce the burden on maintainers. To this end, we introduce a tool to automate rule deduplication that proves effective in 78.7% of cases. In the future, this tool can be integrated with other alerts (e.g., rule convergence and divergence, CVE coverage) into a GitHub bot to provide automatic feedback to contributors.

2 Background

Enterprise Security. EDR and SIEM systems are a cornerstone of modern enterprise security. EDR products generate a stream of endpoint event telemetry, while SIEMs aggregate an organization's security telemetry from a variety of sources including EDR. They overlap in that the product classes can both provide heuristic threat detection capabilities by matching rules against the telemetry stream(s). These rules encode known adversarial behavior as searchable queries that can be run on system logs. Rule-based systems enjoy wide deployment due to their explanability, as an analyst knows exactly which system behavior triggered a alert, and also because rules are frequently annotated with metadata containing additional threat intelligence.

Sigma and Open-Source Threat Detection. Sigma started as a standardization effort for the creation of threat detection rules. Analogous to Yara for malware [71], Snort for network traffic [20], and STIX for threat intelligence [54], the Sigma specification defines a standard, open-source detection format to capture malicious behaviors. An obvious benefit of an universal format is that it enables security professionals across organizations to write and exchange rules. While Sigma describes itself as a generic format for SIEM systems, the rule set heavily focuses on endpoint activity and is largely suitable for EDR. The flexibility of the Sigma format also enables the exchange of various metadata information, such as likely false positive (FP) scenarios. Beyond the format, the Sigma ecosystem has expanded with the release of tools that convert Sigma rules into compatible formats for major security platforms, including Crowdstrike, Splunk, Rapid7, Elastic, IBM QRadar and Datadog at time of writing. Thus the Sigma ruleset itself enjoys tremendous mid-market popularity, especially with SMBs, MSSPs, and MDRs due to its deployability as a free resource. The combined market share of Sigma's known integrations in the SIEM space is 63% [6]. The founding Sigma team maintains the most prominent rule collection as a open-source GitHub repository [62].

We focus on Sigma in this work as it plays a dominant role in the threat detection ecosystem, especially in comparison to other open-source rulesets. Sigma surpasses all other rulesets in both number of rules (3,543) and repository popularity (~10,000 GitHub stars). Aside from Sigma, Elastic (1,503 rules and ~2,600 stars) and Splunk (1,797 rules and ~1,600 stars) are the next most prominent open-source rulesets; notably, both are commercial in addition to being open-source. Other rulesets include Wazuh (655 rules and ~1,300 stars), Panther Labs (858 rules and ~400 stars), and Sublime (1,115 rules and ~300 stars), but their rules are primarily application specific (e.g., cloud services, 2FA, email) and have limited breadth compared to Sigma, Elastic, and Splunk. Further, we calculate that Sigma has coverage of 78% of MITRE ATT&CK v17.1 techniques, in comparison to 72% (Elastic) and 65% (Splunk). While ATT&CK coverage is not an effective metric to assess rulesets [70], this serves as a preliminary confirmation of Sigma's breadth in comparison with commercial products.

Anatomy of a Sigma Rule. A Sigma rule is a YAML formatted document containing several fields. The heart of the rule is in the `detection` field that encodes the malicious behavior of interest. The detection is accompanied by several metadata fields that provide context for a rule when it is deployed. We walk through the detection and metadata fields through an example rule shown in Figure 1. The detection is typically composed of three high-level components: `selection`(s) of malicious behaviors, `filter`(s) of known false positives (FPs), and a `condition` statement of boolean logic that defines how to select and filter. In this example, the rule (in lines 18–25) detects abnormal subprocesses from the Unified Messaging service, associated with potential CVE-2021-26857 exploitation. Therefore, the selection looks for the parent process corresponding to this service: `UMWorkerProcess.exe`. The filter excludes benign subprocesses related to Windows Error Reporting.

The metadata contains high-level descriptive fields (`title`, `description`) and administrative fields that aid in repository management and rule attribution (`date`, `modified`, `id`, `author`). More importantly, the remaining fields enable security professionals to deploy, contextualize, and investigate the resulting rule alerts. For the example, the `logsource` field defines that the rule should operate on Windows logs containing process creation events. The rule has a stable `status`, indicating that the rule has not received negative feedback from the community for an extended period, and a `high` level, indicating a threat of high relevance with rare false positives. The `falsepositives` field does not enumerate any additional FPs. In short, any alerts fired by this rule should be prioritized by a security analyst. For further investigation, the rule also links to a Microsoft blog in the `references` and provides MITRE ATT&CK technique and tactic `tags`. For a summary of the metadata fields, see Appendix Table 5.

3 Dataset

We conduct an analysis of the Sigma threat detection rule repository hosted on GitHub [62], examining its full commit history as of the July 2025 release. Additionally, we collect associated GitHub metadata including contributor GitHub profiles, repository forks, and text from pull requests and issues. We discuss the ethical considerations of our data collection procedure in §12.

In addition to the query itself, a Sigma rule also contains metadata including a description of the attack behavior, a criticality level, and related web references. The most recent release at the time of our data collection consists of 3,543 rules in total. A Sigma release is divided into various rule types and packages [61]. The breakdown of rules by package in the dataset's most recent release is shown in Table 1. The majority

Package	# Rules	# Detect. Edited	# Meta. Edited
Core	1,389	782	1,384
Core+	2,661	1,399	2,654
Core++	2,726	1,413	2,673
Emerging Threats	385	165	360
Other	432	212	417
Total	3,543	1,790	3,450

Table 1: **Sigma Packages** (2025-07 Release): Core++ includes all rules in Core+, which includes all rules in Core.

of rules are released as part of the Core packages, which tend to produce fewer false positives. The three Core* packages are concentric—all Core rules are part of Core+, and all Core+ rules are part of Core++. Approximately 10.9% of rules are part of the Emerging Threats package. The remaining 12.2% of rules are associated with the Threat Hunting type or have lower criticality/status. These are exploratory rules that are not intended to be deployed in production environments.

To provide a commercial baseline in comparison to the crowdsourced Sigma, we briefly examine the Elastic and Splunk rulesets. We retrieve the July 2025 release and full commit history for both datasets. The Elastic ruleset contains 1,503 rules and the Splunk ruleset contains 1,797 rules. Like Sigma, each rule consists of a detection query annotated with various metadata.

4 RQ1: What are the dynamics of rule contribution?

We first aim to understand how individual rules in the Sigma repository evolve over time. We use the GitHub history to analyze how and why rules are added, modified, and removed from the repository. We adapt Vermeer et al.'s tool [69] to compute diffs between each discrete version of a rule from the commits associated with tagged releases.

Longitudinal Changes. The longitudinal activity is shown in Figure 2. Here, we display only modifications to the detection logic itself, rather than a rule's metadata. Metadata modifications, which are much higher frequency, are discussed later in §4. While Sigma does not have a consistent release schedule, we observe a steady growth in total rules over time. Rule deletion is uncommon, excepting two releases in 2019 and 2020 where large numbers of rules are simultaneously added and deleted. However, these are not true deletions, but artifacts of a schema update where rules are deleted and replaced with a reformatted version. In addition to deletion, rules can also be deprecated, meaning the maintainers no longer support or develop the rule. Such rules are moved to a special folder within the repository. However, deprecation consists of only 0.15% of the longitudinal changes, and thus is

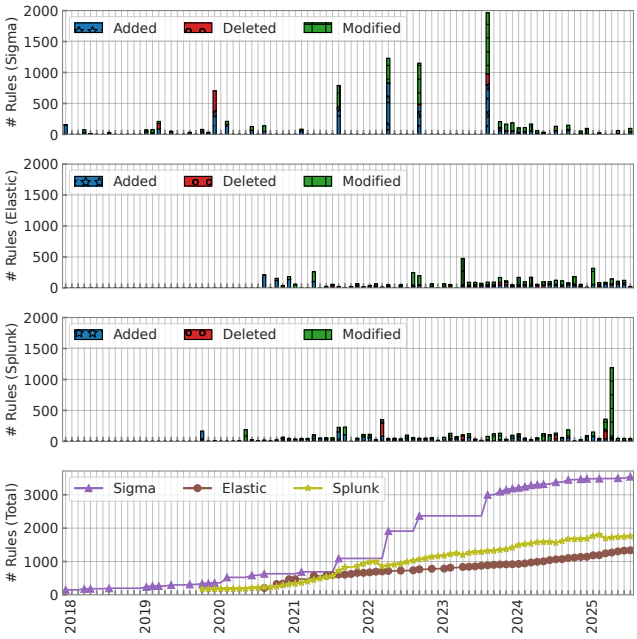


Figure 2: **Longitudinal Rule Modification Activity:** The y-axis of the top 3 subplots indicates the number of rule queries that have been added, deleted, or modified between tagged releases. Only modifications to the detection query are reported (not the rule metadata). If there was more than one tagged release in a month, the deltas are aggregated. The bottom subplot shows the progression of total number of rules per ruleset, with markers indicating months with an official release (e.g., Sigma had two official releases in 2022).

omitted from the plot. The increased rate of rule contribution starting in 2021 has implications for ongoing maintenance. Are the crowdsourced contributors credible? Is it feasible to continue the current maintenance practices as the repository scales? We explore these questions in §5 and §6.

Splunk and Elastic demonstrate similar trends to Sigma with regard to all types of longitudinal changes. Deprecation is associated with less than 0.5% of longitudinal changes for both rulesets and is again omitted from Figure 2. We emphasize that Figure 2 represents *tagged releases* on GitHub, rather than development activity on the repository. We observe that Sigma only moved to an official release cadence of at least once per quarter since mid-2023; Elastic and Splunk have both maintained an approximately monthly release schedule over their development history. Despite this difference in release schedules, Sigma's development branch experiences near daily activity, which is in line with Elastic and Splunk.

Rule Addition. Intuitively, new rules are often added in response to newly discovered threats. Thus, we aim to quantify the responsiveness of the Sigma repository to new vulnerability disclosures and compare to Elastic and Splunk.

CVE Reactivity. We first identify rules that are associated

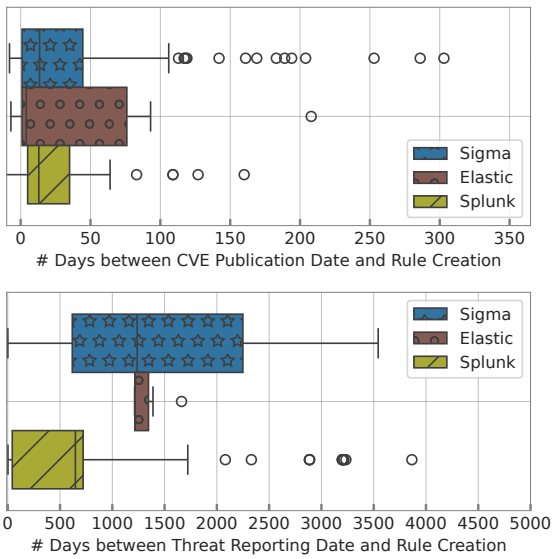


Figure 3: **Comparing Reactivity to CVEs and Other Threats:** All three repositories have a median reaction time within 2 weeks of a CVE's publication. The reactivity to attack campaigns and threat groups in the bottom subplot is much larger due to threat disclosures occuring years before the repository creation.

with CVEs in each ruleset. We retrieve the corresponding CVEs from the MITRE database [67] and compare the rule creation and CVE dates, as shown in the top subplot of Figure 3. We identify 214 Sigma rules that are explicitly associated with 126 CVEs in total. The Sigma repository has a median response time to new CVEs of 14 days. In addition, we observe 28 rules that are created prior to the publication of the associated CVE. 20 of the 28 rules are created within the 2 week period prior to the CVE publication. These are likely caused by an informal disclosure of the vulnerability before official publication. A more extreme case is the rule titled `Malicious PowerShell Commandlets - ScriptBlock` associated with CVE-2021-1675, which was created over four years earlier in March 2017. This rule consists of a series of PowerShell script commands associated with malicious activity and thus is iteratively edited as new attacks surface. Thus while the rule creation date is much earlier than the CVE publication date, the relevant commands (and rule metadata on the CVE) were actually added in January 2023. There are also 23 rules that are created more than a year after the CVE publication date. We observe instances of rule creation due to a resurgence of existing threats. For example, three rules linked to CVE-2014-6287 were introduced in July 2022 due to the discovery of related Apache server vulnerabilities.

In comparison, Splunk has 134 rules associated with 69 CVEs and Elastic has 41 rules associated with 32 CVEs. Splunk and Elastic have median response times to new CVEs of 13 days and 4 days respectively, as seen in Figure 3. We again observe rules that are published prior to the respective CVE due to a likely informal disclosure before the official publication. Both commercial rulesets also have outliers that are created more than a year after the corresponding CVE. Finally, we conduct t-tests between each pair of rulesets with the null hypothesis of equal ruleset CVE latencies. Given p values greater than 0.05 for each pair, we cannot establish a statistically significant difference between the CVE reactivity of any of the three rulesets.

Threat Group Reactivity. In addition to CVEs, we also would like to track response to other threat entities like attack campaigns and threat groups. Following the keyword matching strategy in [70], we identify rules in each ruleset that match threat entities and retrieve the corresponding entries from the MITRE ATT&CK CTI database [7, 9]. While CVEs have a clearly defined disclosure date, attack campaigns and threat groups tend to be longer running in terms of activity and take a longer time for the community to reach consensus. For such threat entities, we define the disclosure date as either the date of emergence described in the text of the ATT&CK CTI description, or the date of publication of the oldest relevant threat report available in the ATT&CK references, whichever is earliest. We compare the rule creation and disclosure dates in the buttom subplot of Figure 3. In the Sigma repository, we identify an additional 56 rules associated with 25 threat entities, resulting in a median response time of 1,238 days. Regarding the commercial rulesets, Splunk has 178 rules associated with 15 threat entities and Elastic has 6 rules associated with 3 threat entities. Splunk has a median response of 647 days and Elastic has a median response of 1,217 days.

In all three rulesets, the median response for threat entities is artificially inflated by a majority of the appearing threat entities being disclosed long before the rule repositories were initiated. In fact, every threat appearing in the Elastic repository was disclosed prior to Elastic's first repository commit. Omitting such threat entities disclosed before Sigma was created results in 14 rules across 9 threats, with a median response of 83 days. Omitting threat entities disclosed before Splunk was created results in 153 rules across 8 threats and a median response time of 43 days. We hypothesize this increased latency in comparison to CVEs is due to the more diverse system behavior attack campaigns and threat groups demonstrate. We do not conduct statistical tests for threat entities due to the number of older threat entities in the dataset.

Rule Modifications. Across all tagged releases, the detection field of 2,088 rules were modified at least once. In the July 2025 release, we find that modifications are slightly more concentrated on the Core package, with 56.3% of Core rules experiencing at least one detection modification. The percentage of modifications drops to 52.6% and 51.8% for Core+ and Core++ respectively.

To better understand these modifications, we conduct a "scope analysis" of changes to the detection field; in other

Overall Scope	Selection Scope	Filter Scope	% Mods
Decreased	No Change	Increased	61.6%
	Decreased	Increased	
	Decreased	No Change	
Increased	Increased	No Change	33.4%
	Increased	Decreased	
	No Change	Decreased	
Unknown	Increased	Increased	5.0%
	Decreased	Decreased	

Table 2: **Rule Modifications:** We find that 61.6% of rule modifications reduce the scope of the rule (i.e., tune out FPs).

words, are rules typically modified to me more permissive (more TP's) or more restrictive (less FP's)? Detection queries are comprised of three components: *selections* that enumerate adversary behavior (Fig. 1, Lines 19-20), *filters* that enumerate false positives conditions (Lines 21-24), and *condition* logic that specifies how to combine the selections and filters (Line 25). We first identified which components within a detection were selections versus filters. By convention, these components are named `selection*` and `filter*`, respectively, but this is not the case for all rules. Instead, we use a Python implementation of lex and yacc (`ply` [16]) to parse the condition logic, generating a truth table that can be used to distinguish selections versus filters. Next, we check the components to identify if any selections or filters have been added or removed. For each selection or filter persisted across the two rule versions, we check the internal subfields (e.g., `ParentImage`, `Image`). For any subfields that have been modified across versions, we calculate (1) if items have been added or removed in lists, (2) the edit distance for modified values within lists, and (3) the edit distance for strings. We also check for the addition or removal of wildcard regular expressions. The results of the scope analysis are presented in Table 2. We see that it is most common for rules to be reduced in scope by removing selections or adding filters (61.6%), compared to 33.4% of modifications that expanded rule scope. `CommandLine` and various `Image` subfields were most commonly modified. Only 5% of modifications could not be procedurally resolved to an overall scope change, e.g., if the selection and filter were simultaneously expanded.

Rule Deletion and Deprecation. Both rule deletion and deprecation are uncommon, consisting of 7.97% and 0.15% of the longitudinal changes respectively. After omitting rule deletions that were a result of schema updates, such rules have an average lifespan of approximately two years (739 days). Rules may be removed because they have detection features that are no longer supported (e.g., alert correlation, non-standard fields) or a newer rule is able to better cover the detection logic. For example, several rules related to the CobaltStrike tool were deleted and combined into one rule that enumerated their logic.

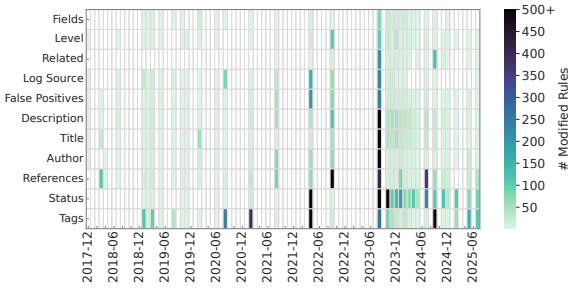


Figure 4: **Modifications to Metadata:** The y-axis is sorted by total modifications, with the most frequently modified field being *Tags*. We exclude fields that do not provide context or interpretive value when deploying rules (e.g., *Date*; changes to this field are largely specification related).

Metadata. The most commonly modified metadata fields are reported in Figure 4. We omit certain fields from the plot that reflect changes to the specification that are not meaningful to rule interpretation or deployment. For example, Sigma's specification update in the September 2024 release alterered the format of the *Date* and *Modified* fields, causing changes in that metadata for over 97% of the rules. Overall, 98% of the rules within the Core++ package have metadata that has been modified at least once.

Some changes to metadata reflect changes in external resources. The *References* field is often updated to remove broken links or to add additional URLs that become relevant as rules evolve. The *Tags* field, which often links to external frameworks like MITRE ATT&CK, MITRE CAR, and the CVE database, is the most frequently modified metadata field. When the ATT&CK framework is updated, tags are updated to reflect the new version. Yet at other times, similar modifications to the *Tags* field actually reflect a change in how the detection should be interpreted. We observed instances where a rule's *Detection* field is updated to capture a behavior that falls under an additional ATT&CK technique, which then caused an associated update to the *Tags* field.

Another field that has been more frequently modified within the last year is *Status*. This field indicates the stability of the rule in deployment, where all rules start as `experimental` and can move up to `test` and then `stable` given no negative feedback from the community. Changes to status tend to follow the order of `experimental` to `test` after approximately 16 months, on average, and `test` to `stable` after an additional 5 months, on average. However, there are 5 rules that go down or back and forth in status, likely because of deployment issues discovered by the community.

Also relevant to deployment is the criticality field, *Level*, which can change in both directions. 64.2% of level modifications are decreases in criticality. This can happen in response to newly identified false positives, as occurred in the modifications associated with pull request `#3587`. The editor identified benign triggers (e.g., changing the brightness on certain Dell

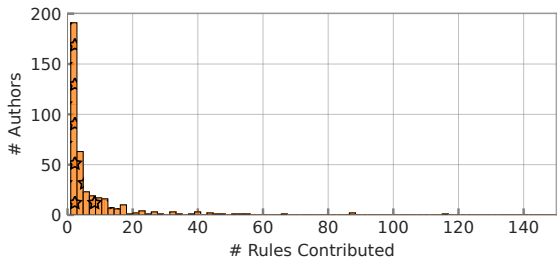


Figure 5: **One and Done Authors:** 37.7% of authors contribute only one rule.

machines) that caused three rules related to Windows Management Instrumentation to fire. In some cases (6.6%), decreases in *Level* provide an alternative to addressing structual limitations in detection logic. In the previous example, the identified false positives were not easily fixable via an added filter, so the level was reduced instead. Here, the *False Positives* field was also updated to reflect the editor's observations. Broadly, *False Positives* is updated based on observations in deployment and as reported by the community.

RQ1: The number of rules in Sigma, with additions of rules far outpacing deletions. Modification of detection logic is widespread, with the most stable rules having been fine-tuned since their introduction. Modifications to metadata are also common, reflecting updates to external resources and providing additional context for interpreting detections.

5 RQ2: Who are the rule contributors and downstream users?

In this section, we delve into the crowdsourced nature of the Sigma project by investigating its contributors and downstream users. Crowdsourcing is a unique aspect of Sigma, as both Splunk and Elastic are developed by their corresponding commercial companies. Therefore we primarily focus on Sigma in this analysis.

5.1 Contributors

Authors. There are 387 rule authors credited in the most recent release of the dataset. The majority of authors contribute very few rules, as shown in Figure 5; 65.6% of authors contribute less than five rules. In contrast, the maintenance team, composed of 5 maintainers based at the company Nextron Systems, is heavily involved. 1,969 (55.6%) of the rules are authored by at least one Sigma maintainer. As a crowdsourced repository, Sigma also has a network of regular contributors outside of the core maintenance team and the one-off authors. We investigate contributors that author rules to understand how their professional background may impact the ruleset. We find users affiliated with several security vendors that contribute rules to the repository, as shown in Table 3. In some cases, credited authors did not interact directly with the Sigma project. Contributors take rules from external sources and list the original rule writer in the author field. Contributors also translate rules from external sources into the Sigma syntax, sometimes incorporating their own information to the original rule. For such rules, both the committer and the original engineer would be listed as authors. For example, developers from Nextron Systems converted 26 rules from Elastic's published detections into the Sigma format. For these rules, a link to the source Elastic rule is listed in the reference metadata. In addition to the committer, Elastic is credited for the rule idea in the author field.

Organization	# Contributors	# Rules Authored	# Rules Edited
Nextron Systems	13	2,124	2,030
Microsoft	13	180	18
HAWK.io	1	66	76
Elastic	3	106	1
Positive Technologies	5	21	45
Zero Networks	2	17	17
Splunk	4	25	1
SCYTHE	1	22	0
Artic Wolf	3	0	17
Kaspersky	2	4	14
Other Organizations	153	887	313

Table 3: **Contributing Organizations:** While the maintenance team authors nearly half of the rules, authors associated with several prominent organizations also contribute to the Sigma repository. The table specifies organizations that offer products related to threat detection. All other organizations are aggregated in the last row.

Editors. Following their initial contribution, rules are iterated upon and edited. Here, we analyze the full commit history, including edits that occur between releases. In addition to being the primary authors, maintainers are also the primary editors for rules. In Figure 6, we observe that only 27.3% of the rules with modified detection logic are not edited by any maintainers. Additionally, 42.1% of rules are only edited by maintainers. In contrast, this also means that over half of the rules are edited by at least one non-maintainer, and further, 15.2% of rules are edited by multiple non-maintainers. This highlights that crowdsourcing also plays a role in rule updates.

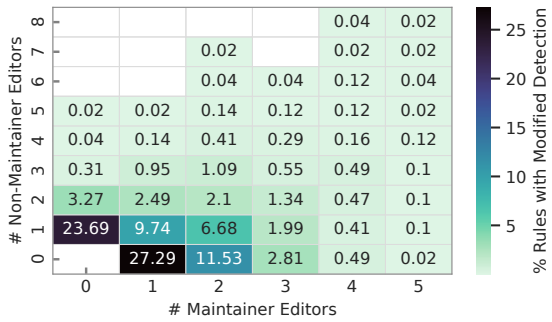


Figure 6: **Crowdsourced Editing:** The heatmap displays how rules with at least one instance of modified detection logic have been edited across maintainers and non-maintainers. While we see that a minority of rules are not edited by any maintainers, non-maintainer editors do play an active role in contributing crowdsourced intelligence. 27.3% of rules are modified by only non-maintainers (the column where maintainers equals 0).

Organizations. In total, we find 127 organizations that contribute to the Sigma repository. 94 of the organizations only have one identifiable contributor. We cannot confirm if contributors are contributing in their official capacity as an organization employee. Regardless, Sigma benefits from the security expertise of these professional analysts and it is likely that these organizations are aware of Sigma as a resource for threat detections. In addition, at least 4 companies that have affiliated contributors explicitly endorse the Sigma project via blogposts and integrations between Sigma and their products [33, 52, 56, 63]. This includes Microsoft's SIEM Azure Sentinel, which is integrated with SOC Prime [52]. SOC Prime is another threat detection platform that whose business model is based in deploying and integrating Sigma rules with established SIEMs, EDRs, and XDRs [63]. In addition, we observe that 6 of the security-focused organizations in Table 3 are contributing to Sigma over multiple years, including Microsoft, Kaspersky, and Splunk.

5.2 Forks

We consider the forked repositories to understand activity from potential downstream users of the Sigma rules. There are 2,379 forks as of the July 2025 Sigma release. We additionally retrieve the GitHub activities (e.g., pushes) for each repository in October 2025 to capture behavior for freshly created repositories; i.e., is a repository created in July 2025 active? Figure 7 compares the fork creation time with the time of the last user push returned by the GitHub API. The user push either corresponds to (1) commit(s) directly authored by the user or (2) commit(s) authored by Sigma contributors on upstream, where the user is passively syncing their repository. These are differentiated by the categorizations in the figure.

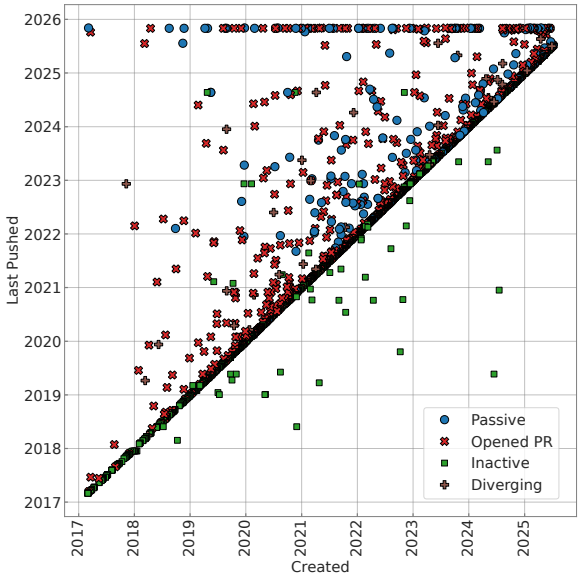


Figure 7: **Fork Activity:** We compare the fork creation time (x-axis) with the time of the last pushed commit (y-axis). Despite the majority of forks being inactive, there is a broader network of Sigma users that passively ingest rules and make their own changes.

We observe four main patterns of activity. 1,594 forks (67.0%) are completely inactive; once the fork is created, the user does not further sync commits from the main repository or push their own diverging commits. These are clustered around the $x = y$ line in Figure 7. We also observe two outliers which have a last pushed timestamp long before the creation time of the fork. Manual inspection revealed that these forks appeared to have been reverted to an earlier commit from the main repository, and had no further activity. In addition, 12 of the inactive forks have a last pushed timestamp more than one day after the creation time, visible above the $x = y$ line in Figure 7. For these forks, there is no visible activity history since initial creation that corresponds to this timestamp, so we still classify them as inactive. We hypothesize that the GitHub API has internally retained the activity timestamps on a subsequently deleted branch.

162 forks (6.8%) correspond to users that may be passively ingesting rules; once the fork is created, the user continues to pull changes from the main repository but does not push their own unique commits. We find that the median duration of the passive ingestion period is 245 days. Further, there is a stable group of downstream users. 22.8% of the passive users have ingestion periods of over a year with the most recent sync of their repositories in 2025.

531 forks (22.3%) correspond to users that have used their fork to open a pull request to the main repository. 255 of these forks (48.0%) have active periods (between the creation and last push time) of less than a month, suggesting that making a one-off contribution to upstream is the primary function

of the fork. However, 130 of these users (24.5%) are active on their fork for over one year. With longer-lived forks, it is possible that these users are also passively ingesting rules.

The remaining 92 forks (3.9%) have diverging changes from the main repository without opening a pull request. The activity period of these forks is skewed, being less than one week for 53 forks (57.6%) and greater than one year for 13 forks (14.1%). Analyzing the diverging changes, we find that 21 users (22.8%) add new rules and 8 users (8.7%) update existing rules in their fork. In addition, 28 users (30.4%) make changes that could correspond to potential deployments—this includes creating or updating configuration files, CI/CD files, backends to convert and/or integrate Sigma rules, and other deployment scripts. The remainder of the diverging changes do not appear to correspond to rule or deployment changes (e.g., minor edits to READMEs). We note that it is possible that these users are still working on an eventual pull request.

We find several organizational accounts among passive, open PR, and diverging users, including DataDog, News Corp, and LinkedIn. Interestingly, these companies are not all in the security industry—for example, News Corp is a media conglomerate. In addition, 117 users opening PRs via forks, 29 passive users and 11 diverging users indicate a career in the security industry in their GitHub profiles. It is possible that Sigma rules may be (1) a component of these organization's security operations, or in the case of security vendors, (2) contribute to their offered detections. We investigate the interchange between Sigma and commercial vendors further in §7. We acknowledge that these public forks do not represent the entirety of Sigma's downstream activity. (GitHub does not have an official mechanism to create private forks of public repositories, but it is straightforward to set this up.) Further, users may be making private changes to rules for their own deployment. However, the forks do establish that Sigma is a resource for commercial vendors and threat detection professionals alike. In contrast, both Elastic and Splunk have smaller communities of visible downstream users on GitHub, with 657 and 457 forks respectively.

> **RQ2:** Sigma has an active community of contributors from other security vendors and the broader industry. Further, there is a stable group of downstream users that passively ingest rules and maintain their own changes. Thus, Sigma functions as a venue for security professionals to iterate upon threat detections.

6 RQ3: What quality control processes are in place?

In previous sections, we have established the presence of a large contributor network and downstream user base, supported by a small maintenance team. We now investigate Pull Requests (PRs) to understand the review process and how it informs the quality of rules in the repository.

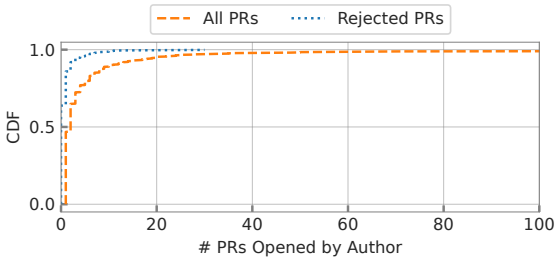


Figure 8: **Pull Request Acceptance:**. The x-axis corresponds to the total and rejected number of pull requests opened by individual authors. Over 60% of authors have no rejected pull requests.

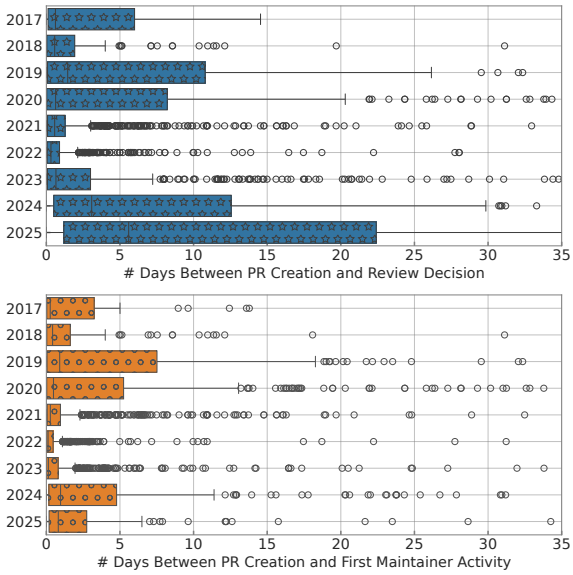


Figure 9: **Time to Pull Request Review:** Maintainers have consistently responded to new pull requests within a day over the course of the repository history.

6.1 Review Process

There are 4,648 pull requests in total, of which maintainers are reponsible for 1729 (37.2%). Unsurprisingly, the top four users that open the most PRs are all maintainers. In Figure 8, we observe a difference in the CDFs for the number of rejected PRs versus all opened PRs per author, especially when the number of opened PRs is less than 10. In other words, repeat contributors are more likely to have to PRs accepted.

Pull requests are primarily reviewed by the maintenance team. However, PRs frequently do not go through through the official GitHub review mechanism, especially early in the repository history—2,904 PRs do not receive official reviews. In total, 1,903 of PRs (40.9%) are merged without review

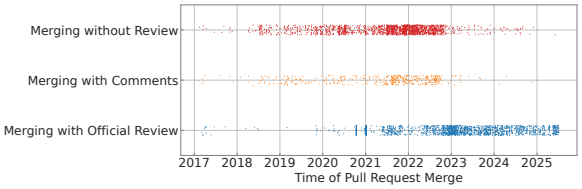


Figure 10: **PRs Through the Years:** Pull requests are either merged without review or comments, merged with only comments, or merged through the official GitHub review mechanism. The x-axis shows the occurrence of each category of pull requests over time. As the repository has grown, the maintainers have more predominantly used the official mechanism for pull request reviews.

or other comment activity and an additional 405 (8.7%) are merged with only comments, as shown over time in Figure 10. Further, 465 PRs (10.0%) are merged by its own (maintainer) author without visible review. It is unclear if these PRs are facing less scrutiny or simply do not have identifiable problems. In other cases, maintainers do leave comments indicating content review. Overall, the repository has moved to a more systematic review process over time. PR review is primarily a conversation between the PR author and maintainer; PR authors make 37.2% of comments and (non-PR author) maintainers make 48.2% of comments on PRs. The remainder of comments are made by other users; 544 PRs (11.7%) receive at least one comment from a broader community member. This is partly due to community events such as OSCD sprints that aim to improve Sigma's coverage of common threats. We also aim to identify if these comments reflect any rule evaluation process. We find that 254 PRs (5.5%) mention evaluation of false positives in the comments.

Figure 9 shows the distributions of the response time to a new PR over the repository history. Aggregating all years, we find that a PR is resolved on average in less than one day, outside of a few outliers. Manual investigation of these outliers reveals that these PRs were abandoned by the original author and eventually closed by a maintainer. The turnaround to the first maintainer response on a PR is even shorter, demonstrating that maintainers respond quickly even if a pull request may take several days to resolve. The breakdown by year in Figure 9 demonstrates that maintainers have retained this speed over nearly eight years of development. We observe that the number of days to review decision has higher variability over different years than the time to first maintainer activity. In particular, 2024 and 2025 have larger upper quartiles (12 and 22 days respectively) than previous years. This is partly credited to the project moving to a model of multiple maintainers needed to approve pull requests, which induces additional latency in the review process.

This is an encouraging sign for Sigma's development. As a basis for comparison, both Elastic and Splunk have defined

Reasoning	Num. PRs	Example
Closed by PR author	262	#3777 [4]
Repository management	151	#4147 [2]
Overlapping or duplicate rule	65	#2916 [1]
Rule quality	29	#700 [3]
Additional discussion required	11	#2502 [5]
Total	518	

Table 4: **Reasons for Rejection:** The majority of rejected PRs are due to low-level reasons rather than rule quality (examples linked in the bracketed citations).

review processes on GitHub, since (1) the contributor and reviewers are from the same maintaining organization and (2) they have larger development teams dedicated to their rule repositories compared to Sigma. Elastic requires approvals from two other developers for any pull request and Splunk requires an approval from a *code owner*, which is a GitHub-defined admin role that appears to correspond to more senior Splunk developers.

6.2 Rejected PRs

We analyze the 518 rejected pull requests to gain further insight into the quality control process. We categorize the reasons for rejection in Table 4.

Closed by PR Author. 79 PRs (15.3%) are closed by the PR author explicitly due to an self-identified error. For example, in `#3777`, the author opens a PR editing a rule to address false positives observed in their own environment. They later close the PR upon realizing this was caused by some other issue. An additional 183 requests (35.3%) are closed by the PR author without an commented explanation.

Repository Management. 151 PRs (29.2%) are rejected for reasons related to repository management. This includes automated PRs opened by bots (e.g., GitHub's dependabot), abandoned PRs, and those that are not compliant with Sigma's PR or rule specifications (e.g., `#4147`). In addition, PRs are also closed and reopened as a new request—either due to logistic errors (e.g., too many merge conflicts, excessively large PR, accidental duplicate PRs), or because multiple contributors have simultaneously opened similar PRs. These rejections are not reflective of the semantic value of the PR content.

Overlapping or Duplicate Rule. 65 PRs (12.5%) are rejected because they attempt to add rules that duplicate or heavily overlap with already existing rules, as in `#4634`. Here a maintainer mentions relying on their own knowledge of the repository ruleset to determine if rules are duplicates and that "there isn't a "good" way to know for sure." These PRs correspond with 108 rules, in total.

Rule Quality. 29 PRs (5.6%) are rejected due to rule quality. This includes proposed rules that lack sufficient selection

conditions, fire excessive false positives, or have practical performance issues. For example, `#619` adds a rule related to spearphishing, but the selection conditions are too vague to fire on any true positive events. In `#700`, the proposed rule is rejected due to a high false positive rate; the rule selects for a tool command (wmic get) that is often benignly used by administrators and software. In contrast, `#1324` is rejected because the detection uses a highly "performance intensive search".

Additional Discussion Required. 11 PRs (2.1%) indicate offline discussion or that additional discussion is required. For example, `#2502` outlines a potential new feature for Sigma rules that requires additional discussion.

Overall the majority of PRs are rejected due to self-filtering by the PR author. At times, this is due to the author having spotted an issue with their rule (79, 15.3%), but most PRs are closed without comment (183, 35.3%). Repository management issues are the next leading cause of PR rejection (151, 29.2%), including PRs that do not properly follow Sigma formatting and specification standards. These rejections reflect low-level issues that may be attributable to crowdsourcing, as some authors are not as experienced in Sigma development practices. However, nearly as frequent as repository management issues are high-level rationales related to quality control (105, 20.3%), including rule duplication (65, 12.5%), quality (29, 5.6%) and the need for deeper discussion (11, 2.1%).

> **RQ3:** The Sigma repository is moving towards a more systematic review process, especially within the last two years. While rejected rules are mostly due to low level reasons, we find that new rules do go through an evaluation process. However, this quality control often relies on the institutional knowledge of a small group of maintainers (e.g., in filtering duplicate rules).

7 RQ4: How does Sigma compare to major commercial systems?

After understanding the dynamics within Sigma itself, we turn to comparisons between Sigma and *commercial* vendors that publish an open-source version of their rules. Do common rules receive similar ongoing maintenance?

7.1 Rule Comparisons

For this analysis, we investigate rules in Sigma that have been authored by at least one contributor associated with Splunk or Elastic, as we hypothesize that these rules are likely to appear in the commercial vendors' repositories as well. From each rule, we extract the fields and values. We take these values and iterate through the histories of the corresponding commercial repository to look for potential rule matches. We manually review any rules that have matching values and verify which rules have common ancestry or converge similarly. Our final set of rules for analysis consists of 55 rules from Elastic and 16 rules from Splunk.

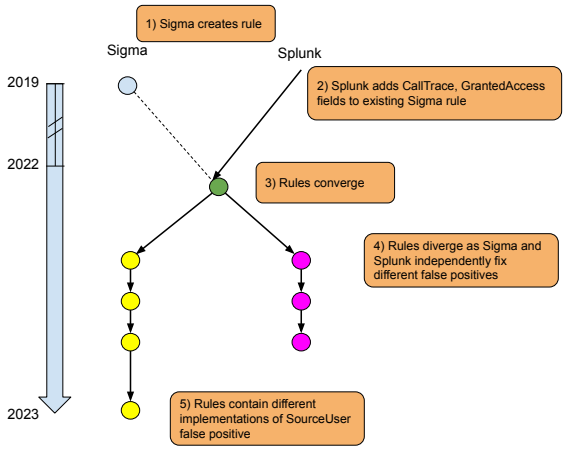


Figure 11: **Convergence and Divergence of a Credential Dumping Rule:**. These rules from Sigma and Splunk converge in 2022 due to a contribution from a Splunk developer to Sigma. Since then, the two repositories have independently made edits to their versions of the rules, causing divergence.

First, we find that 64.8% of rules with common authors between Sigma and a commercial vendor do not have semantic differences (we exclude syntactic changes that are specific to individual rulesets). Of the remaining 35.2%, we find that the changes are primarily false positive refinement. In this section, we explore some examples of diverging rules as case studies to characterize the differences and what may be causing them. More cases can be found in Appendix A.

Passive Crowdsourcing. First, we note that 11 instances (15.5%) of divergence between Sigma and Elastic have a distinct pattern of crowdsourcing. That is, Elastic does not actively contribute their rules. Rather, an non-Elastic author uses an Elastic rule as a reference to build a Sigma version. Further, the Sigma rule integrates detection components from other references, causing a divergence from the Elastic rule at the time of creation. An example can be seen in Appendix A.

A Case of Convergence and Divergence. We investigate a pair of rules from Sigma and Splunk addressing the same adversary behavior that have converged and since diverged, as shown in Figure 11. The Sigma rule on credential dumping with `lsass.exe` was added to the repository in 2019. In 2022, Splunk independently wrote a rule on this behavior, with more values specified in the `GrantedAccess` and `CallTrace` fields. The Splunk develop also opened a pull request for these changes on top of the existing Sigma rule. This change meaningfully expanded the scope of the Sigma rule detection, so the Splunk developer was added as an author in the Sigma repository. At this point, the Sigma and Splunk rules were identical in function. Since then, both repositories

have independently made changes related to selection criteria and false positives, causing divergence in the rules.

The most recent versions of this rule are shown in Appendix A. Notably, one of the shared false positive (FP) conditions filters out events that have a `SourceUser` related to the LocalSystem account on Windows—this is usually the most privileged account on a Windows machine [46]. Splunk explicitly searches for the associated token `NTAUTHORITY\\SYSTEM`, while Sigma instead uses a string search for `SourceUsers` containing "AUTHORI" or "AUTORI". Splunk also includes the token for the Windows NetworkService account `NT AUTHORITY\\NETWORK SERVICE`. Here Sigma's more flexible implementation of the FP opens the potential for an adversary to evade this detection rule. We also observe differences in the `GrantedAccess` selection. The Sigma commit messages indicate some of these values were removed due to excessive false positives; however, this information is not propagated back to the commercial vendor. In addition, the Sigma version of the rule has some false positive filters grounded in data from rule deployment. This speaks to one potential benefit of crowdsourced repositories. They provide a unified location to report FPs and intelligence.

The Impact of Crowdsourcing. In another case, we observe divergence in the Sigma repository from the commercial vendor due to multiple organizations contributing to the rule (shown in Appendix A) related to the deletion of shadow copies. In fact, the commercial vendor version of this rule detects only a subset of the `selection1_img` events detected by the Sigma rule. A developer from the commercial vendor was one of the original authors of the Sigma rule. Authors affiliated with other organizations have since contributed, including the oscd.community effort. Similarly to the previous example, we find that these organizations provide additional detection logic that does not integrate back into the commercial vendor's version. While it is positive in this situation, we acknowledge that this is also a risk of crowdsourced security. Particularly when new contributions are not comprehensively vetted, ineffective detections can have widespread downstream effects.

Production Versus Deprecation. Another interesting example is a rule related to privilege escalation (see Appendix A), where the actual detection logic is consistent between Elastic and Sigma. This rule originated from Elastic. A few months after it was added to Sigma, the rule was deprecated in the Elastic repository. However, the deprecation did not propagate back to Sigma. Interestingly, the Sigma version continues to define the rule as `high` criticality.

RQ4: Crowdsourced threat detection offers both strengths and weaknesses as compared to commercial vendors. Crowdsourcing enables iteration on vendor-provided rules and benefits from widespread identification of FP sources. However, in at least one case, we find Sigma's overly-permissive FP filter can permit evasion.

8 Disclosure

We begin by sharing the remarks of one of the Sigma maintainers from a conversation in which we presented our results.

Repository Trends. Over the past two years, the Sigma maintainers have taken steps to improve the maturity and reliability of the repository. One of the biggest changes was organizing the project into releases in which rules are grouped by their "cost" of use. While Sigma had initially assumed that users would review and fine-tune rules before deployment, they have found that many users have assumed that all rules were ready to deploy out-of-the-box. This led to complaints that Sigma generates too many False Positives, especially in small- to medium-sized organizations that lack the resources to investigate a large number of alerts. To meet users where they were at, Sigma formalized the release system depicted in Table 1. This tiered system directs users with limited security budgets to the most mature and deployment-agnostic rules while still providing more comprehensive detection and threat hunting rules for better-resourced organizations. The release system also reflects one goal of the Sigma project – rather than have as many rules as possible, prioritize the quality and efficacy of rules. The tiered releases expose a pipeline in which rules, as they mature, can move from nominal coverage — perhaps with high false alarms — to confirmed coverage.

Community. An interesting distinction between Sigma and other threat detection engineering efforts is that Sigma's rule development is not driven by telemetry — instead, they rely on the insights of their community. Broadly, the maintainers find that most Sigma contributors do not have a long history of experience with the project. Most new contributors share just a handful of rules and then do not return. That said, it is also the case that downstream Sigma users are often inexperienced security practitioners. For example, small- to mid-size MSSPs and MDRs often incorporate Sigma into their services. Such businesses need rulesets that provide broad coverage of threat behaviors, but often lack the resources for extensive internal development of proprietary rules. However, in spite of contributions from several major security vendors, there exist no formal relationships between Sigma and these businesses.[1] To grow the community, the Sigma maintainers are continuing to develop tools for converting Sigma rules into

[1] A maintainer mentioned that, anecdotally, they are aware of Sigma rules that have found their way into major vendors' products.

other major security platforms. making it a popular choice for the managed service market.

Rule Review. In terms of review of new rules, Sigma's idealistic goal is to have four contributors approve the rule before its inclusion. However, its realistic goal is to have at least two contributors review, and at times a maintainer may approve a rule themself if they possess sufficient expertise in the area. The maintainers' review process includes checking rule submissions against Virus Total and even manually attempting to replicate the attack, where possible. The Sigma maintainers play an important role not only in rule creation, but also in editing contributor-proposed rules. It is often the case that contributors may take weeks to respond to comments on a pull request, forcing maintainers to resolve issues themselves.

9 Operationalizing Our Findings

Overall, our findings paint a favorable picture of crowdsourced threat detection engineering. Sigma's coverage of threats is steadily increasing, and the project reacts quickly to new threats (CVEs). In both respects, Sigma's performance is on par with major industry-backed public rule sets (Splunk, Elastic). Sigma also possesses unique benefits as compared to these products. In direct comparisons to CVE coverage, Sigma was able to identify unique false positive conditions that Splunk did not. More broadly, the proportion of rule edits that reduce rule scoping (61.6%) suggest that this is a core strength of the crowdsourcing approach. Further, unlike other public rulesets, Sigma benefits from immediate compatibility with major backend platforms.

That said, our analysis also uncovered potential points of fragility. In spite of being an essential service for many downstream stakeholders, Sigma has not (yet) developed a community of independent regular contributors beyond the maintainers. As a result, the typical experience level of independent contributors is low. This places added pressure on the PR review system to ensure that proposed changes or additions actually improve the ruleset. Unfortunately, the formal review system is already facing difficulties of scale, with PRs often being reviewed by fewer than 4 contributors. The specifics of rule review require a great deal of both institutional knowledge and professional expertise on the part of the maintainers—encyclopedic knowledge of existing rules is required to prevent duplicated entries, while time-consuming manual effort may be required to reproduce vulnerabilities for rule testing. In the remainder of this section, we will discuss (1) our automated methodology to improve the Sigma review process and (2) future interventions that may be implemented.

9.1 Automating Rule Deduplication

As seen in §6.2, maintainers rely on their knowledge of the entire ruleset to determine if incoming rules are duplicate or overlapping with the existing rules. Not only does this increase the barrier to entry for new maintainers and contributors alike, but also it poses scaling problems as the repository continues to grow. Thus, this process can benefit from automation. We adapt our code from §4 for characterizing rule modifications to instead detect duplicate or overlapping rules. As discussed in that section, we can broadly summarize the differences between two rules as (1) added or deleted selections and filters, (2) additions or deletions from lists and (3) modifications to values like strings. We calculate a distance function by summing the number of added or deleted fields and values with the edit distances of modified values scaled to a value between 0 and 1. Intuitively, a candidate rule can be compared to every rule in the existing ruleset to identify if there are any rules that it is "close" to. When we test a candidate rule that is truly overlapping with existing rules, this results in a unimodal distribution with outliers or a bimodal distribution where the two peaks represent the group of similar rules and the rest of the ruleset. Thus we can use the derivative of the distribution function to automatically set a threshold that results in a output group of similar rules that already exist in the ruleset.

We utilize the 108 rules identified as overlapping or duplicate by maintainers (corresponding to 65 pull requests) in §6 as a real world, ground truth dataset. We are able to effectively identify the duplicate or overlapping rule within the output set for 85 of the 108 rules (78.7%). We augment this dataset by randomly generating perturbations of 100 rules from the latest Sigma release for further evaluation. We generate perturbations by randomly choosing one field within a rule and depending on its type, adding or deleting an element with equal probability, or randomly modifying its value. We are able to identify the original rule for 91 perturbations. On average, the output group contains 8 rules for our real world dataset. We emphasize that this still drastically reduces the manual burden on maintainers to identify potential rule duplicates, given that the full ruleset contains over 3,000 rules. We also test the methodology on a random sample of 50 rules from the latest release to identify if any false duplicates would be flagged, resulting in a false positive rate of 16%.

9.2 Future Interventions

Given our findings, we believe that mechanisms for (semi-) automated evaluation of threat detection rules would greatly amplify community efforts. Aside from rule deduplication (§9.1), other low-level review tasks can be automated. For example, in §7.1, we note a Sigma rule whose overly permissive check of the `SourceUsers` field permits evasion. Since regular expressions may be permissible in some selection criteria, it should be possible for a machine learning model to identify general rules for variable assignments. Further, automation of the synchronization between open source repositories can augment the quality control process. Alerting when rules across repositories converge or diverge provides additional context

for rule updates. Such alerts can be implemented as a GitHub bot to streamline the process for maintainers.

Put more broadly, our findings highlight the potential impact for automated threat detection rule evaluation research. This is not to undercut the importance of the Sigma community's peer feedback loop—hands-on testing across many environments will continue to be essential for identifying both missed detections and false positive sources—but partial automation will help the community to make better use of available expertise. Along similar lines, we also see research opportunities for contributor training and onboarding. Like any open-source project, Sigma would benefit from a broader network of experienced contributors; how can we most effectively close the knowledge gap between experienced and inexperienced contributors? There are two kinds of inexperience to consider: lack of experience as (1) a Sigma contributor, or (2) a security practitioner. In the former case, automation efforts could form an instant feedback loop to help practitioners quickly address issues in their rule submissions. The latter case requires studying how interventions and trainings can best onboard inexperienced practitioners. To this end, Sigma has employed community sprints to positive effect, although these have relied on a mix of seasoned rule developers and mentors. Future work may empirically explore the effects of different training methods on engineering efficacy.

10 Discussion

We note the following limitations of our methodology. First, GitHub activity provides only partial visibility into an open-source project community. Rationales for decisions are not always made explicit in the GitHub history; for example, we observed occasionally references to offline discussions in pull request comments, and the majority of author-closed pull requests do not include a commented explanation. We are likewise unable to observe the full reach of Sigma on GitHub due to private forks and clones, etc.; our analysis of community size is therefore an underapproximation.

We note that empirical evaluation of individual rule performance is out of scope for the purposes of our study. Broadly this is a challenging problem because the discriminatory power of a rule is not only influenced by the temporal dynamics of an evolving adversary, it is also deployment-specific. A rule that effectively detects threats with high signal-to-noise in one environment may generate many false positives in another. This is an innate limitation of the rule-based threat detection model currently dominating industry.

Our study is also not positioned to speak to the representativeness of Sigma among crowdsourced threat detection engineering efforts. To the best of our knowledge, Sigma is singular in terms of its scale and buy-in, but other worthwhile communities may exist. Likewise, we do not provide comparison to related but orthogonal efforts such as crowdsourced antivirus signatures and threat intelligence. Additionally, threat detection rules can be used in both SIEM and EDR products, but are more essential to the value proposition of EDR. It is likely not a coincidence, then, that the industry-backed public rulesets we consider are both SIEM, while EDR rulesets are proprietary. It may be the case that EDR rulesets are held to a higher level of rigor in terms of false-positive tuning or CVE reactivity. Finally, we note that our early study plans considered the threat of *poisoning attacks*, but this idea was later discarded. Malicious rules may generate more false alerts, but do not invalidate the coverage of existing rules. Further, only the most stable and effective rules are admitted to Sigma's core package, allowing time for missed detections or false positives to be reported.

Overall, our study fits into the existing body of work on open-source and crowdsourced software. Well-known open source projects such as Apache [49] and FreeBSD [23] historically demonstrate similar trends as Sigma, with a small team of core developers contributing to the majority of the codebase, and many others [65] have a community of external contributors that see their pull requests rejected for similar reasons as in Section 6.2. One aspect of the Sigma ecosystem that we observe is that its crowdsourcing is often occuring from security experts, differentiating it from crowdsourcing studies where the contributors are primarily laypeople (in the context of security) or end users. Further, while evaluating the validity of crowdsourced contributions is a common requirement across security and software domains alike, our work considers the unique aspects of threat detection rules. As previously discussed, evaluating threat detection rules is a complex task due to its potential dependencies on the deployment environment; this poses additional challenges for a crowdsourced project.

11 Related Work

Open Source and Crowdsourced Security. Crowdsourcing has been studied in the context of many security-related domains, including phishing [40, 50, 53], vulnerability and malware investigation [11, 14, 15, 51, 75], and web and security telemetry [8, 10, 35]. While many of these studies focus on crowdsourcing as a methodology for large-scale data collection, another body of work investigates the implications of open source and crowdsourced development (or resources) on general software security [42, 45, 74]. The infamous "hypocrite commits" incident in the Linux kernel codebase [44] resurged awareness on the threat model of malicious open-source contributors; Wermke et al. [74] identified that open source maintainers were aware of the risk of malicious contributors, but had not dealt with such incidents themselves.

One line of work has taken the supply chain security perspective [39] and investigated the downstream effects of open-source developers' practices on security [13, 37, 38, 73]. Similar to findings from [37] on OSS developers' threat modeling, we find that the broader Sigma contributor community ex-

hibits ad hoc development behavior for threat detection rules. Given that the mitigations for improved threat modeling versus improved rule development are different, it is important that both these ecosystems are studied in depth. Another class of papers considers development scenarios where the project itself is security focused [28, 30, 51] such as bug bounty programs [22] and threat intelligence [18, 55]. Additionally, investigations of how open-source and crowdsourced project structures affect software development have also been conducted outside of a security context [12, 23, 49, 57, 65, 66]. While our findings share some high level commonality with these studies (e.g., similar contribution patterns from the maintenance team), our focus on threat detection rules results in a distinct threat model in comparison to generically open-source software.

Threat Detection and Threat Intelligence. While Sigma is a repository of threat detection, prior academic work has frequently focused on analyzing collections of threat intelligence [17, 31, 32, 43]. Geras et al. [31] conduct an interview study to understand how the quality of threat intelligence is assessed and improved; while qualitative, their aims are analogous to our work on threat detection. Regarding threat detection rules, prior work has considered the development of network intrusion detection system (NIDS) rules [29, 69]. Our work is closest to Vermeer et al. [69], who conduct a longitudinal analysis of commercial and open source NIDS rules deployed by an MSSP. In contrast, Sigma provides rules to be run on an endpoint. Prior work on endpoint detection has considered performance of systems in specific simulated or real world attack scenarios [36, 60], provenance-based methods [24], and integration of rules with MITRE ATT&CK [70]; analysis of rule development has largely not been considered. A recent work by Uetz et al. [68] introduces a system to generate evasions against threat detection rules, which they evaluate on the Sigma Windows process creation rules. This is one promising direction towards automating rule evaluation.

12 Conclusion

We present an in-depth exploration of the Sigma ruleset towards understanding how crowdsourced threat detection is developed. We find that the consistent growth in the Sigma repository is supported by a iterative loop of false positive tuning and a broader community of security professionals. We observe that the review process involves evaluation of rule quality, although there is reliance on institutional maintainer knowledge. Finally, comparative case studies of overlapping rules in Sigma and commercial vendors reveal the strengths and weaknesses of crowdsourcing security. We conclude with a discussion of our disclosure to the Sigma maintenance team and how to operationalize our findings. We introduce a tool to partially automate the Sigma review process by identifying duplicate rules and make additional suggestions towards improving the development of threat detection rules.

Acknowledgments

We thank the reviewers for their helpful suggestions. This work was supported in part by NSF grants 2055127 and 2229876.

Ethical Considerations

Stakeholders We enumerate the stakeholders involved in this research. First, we have our research team and the stakeholders that are directly impacted by our data collection: those that interact with the Sigma repository via Github– this includes the maintenance team as well as GitHub users that have made commits, pull requests, comments, or forks of the repository. In addition, there are parties that are indirectly affected by this work. Sigma has a larger downstream user-base than captured in their GitHub data, such as organizations and users that deploy Sigma rules or utilize the Sigma project as a reference. Broadly, threat detection vendors, users, and researchers are all stakeholders in this work.

Impacts and Mitigations First, we discuss the impacts on our stakeholders directly impacted by our data collection (i.e., the GitHub users captured in the dataset) and how we mitigate potential harms to them. Prior to the study, we submitted our intended data collection (i.e., GitHub repository commits and metadata, including contributor profiles on GitHub) to our Institutional Review Board; the study was assessed as exempt. While all of our collected data is publicly available on GitHub, collecting personal data about contributors is a potential risk. Thus, we aim to minimize the information extracted about individual contributors. We are primarily interested in contributors' professional affiliations and experience categorically (e.g., works for a security vendor) and do not seek to collect further personal information. We use the GitHub profiles to extract a mapping between GitHub username to the user's professional organization and industry–if and only if the user has provided this information in their profile. We do not consider any other profile data for analysis. Further, we will not include the username to organization mapping in the Open Science release. With these procedures, we aim to minimize potential harm to individual contributors, while gaining insight into the professional backgrounds of the Sigma community. Specifically, collecting this organizational mapping data allows us to draw conclusions about the impact and extent of the Sigma project. Abstractly, this also enables us to understand how the training process of rule development may be improved in the future. In addition, we have disclosed our results to Sigma and introduced tools to mitigate certain issues in the review process that we identified. Improving the Sigma ecosystem has downstream benefits for the security of the organizations that deploy and reference the project.

Decision to Conduct Research This work aims to characterize the crowdsourced threat detection project Sigma and

introduce improvements into its rule development process. Given the open source nature of our data and the precautions mentioned above, we believe that our work poses minimal risk to the contributors captured in our dataset. At the same time, we introduce tools and insights that provide benefit to the Sigma project as well as its downstream user base. Further, making our tools and insights public can also help the security community at large conduct future research related to threat detection.

Open Science

We provide a link to our code and data at https://doi.org/10.5281/zenodo.20437918. Instructions on how to set up and run the code are provided in the README. As discussed in §12, we omit data collected from contributor profiles (and corresponding analysis) in this release to minimize risks to contributor privacy.

References

[1] Create file_event_win_exchange... https://github.com/SigmaHQ/sigma/pull/2916, 2024.

[2] Create win_security_authentication_with_valid_users.. https://github.com/SigmaHQ/sigma/pull/4147, 2024.

[3] Create win_wmi_local_recon.yml. https://github.com/SigmaHQ/sigma/pull/700, 2024.

[4] Fix false positive on proc_creation_win_proc_wrong. https://github.com/SigmaHQ/sigma/pull/3777, 2024.

[5] Introducing macros. https://github.com/SigmaHQ/sigma/pull/2502, 2024.

[6] Best siem software in 2026. https://6sense.com/tech/security-information-and-event-management-siem, 2026.

[7] Campaigns. https://attack.mitre.org/campaigns/, 2026.

[8] Greynoise intelligence. https://www.greynoise.io/, 2026.

[9] Groups. https://attack.mitre.org/groups/, 2026.

[10] Sans internet storm center. https://www.dshield.org/, 2026.

[11] Omer Akgul, Taha Eghtesad, Amit Elazari, Omprakash Gnawali, Jens Grossklags, Michelle L Mazurek, Daniel Votipka, and Aron Laszka. Bug {Hunters'} perspectives on the challenges and benefits of the bug bounty ecosystem. In *USENIX Security Symposium*, 2023.

[12] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. Why does code review work for open source software communities? In *International Conference on Software Engineering*, 2019.

[13] Sabrina Amft, Sandra Höltervennhoff, Rebecca Panskus, Karola Marky, and Sascha Fahl. Everyone for themselves? a qualitative study about individual security setups of open source software contributors. In *IEEE Symposium on Security and Privacy*. IEEE, 2024.

[14] Jessy Ayala, Steven Ngo, and Joshua Garcia. A deep dive into how open-source project maintainers review and resolve bug bounty reports. In *IEEE Symposium on Security and Privacy*, 2025.

[15] Jessy Ayala, Yu-Jye Tung, and Joshua Garcia. A {Mixed-Methods} study of {Open-Source} software maintainers on vulnerability management and platform security features. In *USENIX Security Symposium*, 2025.

[16] David Beazley. Ply (python lex-yacc). https://ply.readthedocs.io, 2025.

[17] Xander Bouwman, Harm Griffioen, Jelle Egbers, Christian Doerr, Bram Klievink, and Michel Van Eeten. A different cup of {TI}? the added value of commercial threat intelligence. In *USENIX Security Symposium*, 2020.

[18] Xander Bouwman, Victor Le Pochat, Pawel Foremski, Tom Van Goethem, Carlos H Gañán, Giovane CM Moura, Samaneh Tajalizadehkhoob, Wouter Joosen, and Michel Van Eeten. Helping hands: Measuring the impact of a large threat intelligence sharing community. In *USENIX Security Symposium*, 2022.

[19] Broadcom. Carbon Black EDR. https://www.broadcom.com/products/ carbon-black/threat-detection-and-response/endpoint-detection-and-response, 2025.

[20] Cisco. Snort. https://www.snort.org/, 2025.

[21] Cisco Talos Intelligence Group. PhishTank. https://www.phishtank.com/, 2025.

[22] Alejandro Cuevas, Emma Hogan, Hanan Hibshi, and Nicolas Christin. Observations from an online security competition and its implications on crowdsourced security. *arXiv preprint arXiv:2204.12601*, 2022.

[23] Trungh Dinh-Trong and James M Bieman. Open source software development: A case study of freebsd. In *International Symposium on Software Metrics*, 2004.

[24] Feng Dong, Shaofei Li, Peng Jiang, Ding Li, Haoyu Wang, Liangyi Huang, Xusheng Xiao, Jiedong Chen, Xiapu Luo, Yao Guo, et al. Are we there yet? An Industrial

Viewpoint on Provenance-based Endpoint Detection and Response Tools. In *ACM Conference on Computer and Communications Security*, 2023.

[25] Elasticsearch B.V. Detection Rules. https://github.com/elastic/detection-rules, 2025.

[26] Feedly. Feedly threat intelligence. https://feedly.com/threat-intelligence, 2025.

[27] Filigran. OpenCTI. https://opencti.io, 2025.

[28] Felix Fischer, Yannick Stachelscheid, and Jens Grossklags. The effect of google search on software security: Unobtrusive security interventions via content re-ranking. In *ACM Conference on Computer and Communications Security*, 2021.

[29] Hugo Gascon, Agustin Orfila, and Jorge Blasco. Analysis of update delays in signature-based network intrusion detection systems. *Computers and Security*, 30(8), 2011.

[30] Thomas Geras and Thomas Schreck. Sharing communities: The good, the bad, and the ugly. In *ACM Conference on Computer and Communications Security*, 2023.

[31] Thomas Geras and Thomas Schreck. The big beast to tackle: Practices in quality assurance for cyber threat intelligence. In *International Symposium on Research in Attacks, Intrusions and Defenses*, 2024.

[32] Harm Griffioen, Tim Booij, and Christian Doerr. Quality evaluation of cyber threat intelligence feeds. In *Applied Cryptography and Network Security*, 2020.

[33] HarfangLab. Perks of sigma and yara rules in an edr. https://harfanglab.io/blog/product/perks-sigma-yara-edr/, 2024.

[34] Jan Hruska and Peter Lammer. Virus bulletin. https://www.virusbulletin.com/, 1989.

[35] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2020.

[36] George Karantzas and Constantinos Patsakis. An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors. *Journal of Cybersecurity and Privacy*, 1(3), 2021.

[37] Harjot Kaur, Carson Powers, Ronald E. Thompson III, Sascha Fahl, and Daniel Votipka. "threat modeling is very formal, it's very technical, and also very hard to do correctly": Investigating threat modeling practices in open-source software projects. In *USENIX Security Symposium*, 2025.

[38] Agata Kruzikova, Jakub Suchanek, Milan Broz, Martin Ukrop, and Vashek Matyas. What johnny thinks about using two-factor authentication on github: A survey among open-source developers. In *International Conference on Availability, Reliability and Security*, 2024.

[39] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. Sok: Taxonomy of attacks on open-source software supply chains. In *IEEE Symposium on Security and Privacy*, 2023.

[40] Daniele Lain, Kari Kostiainen, and Srdjan Čapkun. Phishing in organizations: Findings from a large-scale and long-term study. In *IEEE Symposium on Security and Privacy*, 2022.

[41] LevelBlue, Inc. Open Threat Exchange. https://otx.alienvault.com/, 2025.

[42] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *ACM Conference on Computer and Communications Security*, 2017.

[43] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Reading the tea leaves: A comparative analysis of threat intelligence. In *USENIX Security Symposium*, 2019.

[44] Linux Foundation Technical Advisory Board. Report on University of Minnesota Breach-of-Trust Incident. https://lwn.net/ml/linux-kernel/202105051005.49BFABCE@keescook/, 2021.

[45] Andrew Meneely and Laurie Williams. Secure Open Source Collaboration: An Empirical Study of Linus' Law. In *ACM Conference on Computer and Communications Security*, 2009.

[46] Microsoft. Localsystem account. https://learn.microsoft.com/en-us/windows/win32/services/localsystem-account, 2025.

[47] MISP project. MISP Open Source Threat Intelligence Platform. https://www.misp-project.org/, 2025.

[48] Audris Mockus, Roy T Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *International Conference on Software Engineering*, 2000.

[49] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *Transactions on Software Engineering and Methodology*, 11(3), 2002.

[50] Tyler Moore and Richard Clayton. Evaluating the wisdom of crowds in assessing phishing websites. In *Financial Cryptography and Data Security*, 2008.

[51] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. Understanding the reproducibility of crowd-reported security vulnerabilities. In *USENIX Security Symposium*, 2018.

[52] Cristhofer Munoz. Azure sentinel sigma and soc prime integration (part 1): Convert sigma rules to azure sentinel. https://techcommunity.microsoft.com/blog/microsoftsentinelblog/azure-sentinel-sigma-and-soc-prime-integration-part-1-convert-sigma-rules-to-azu/1232903, 2020.

[53] Hiroki Nakano, Daiki Chiba, Takashi Koide, Naoki Fukushi, Takeshi Yagi, Takeo Hariu, Katsunari Yoshioka, and Tsutomu Matsumoto. Canary in twitter mine: Collecting phishing reports from experts and non-experts. In *International Conference on Availability, Reliability and Security*, 2023.

[54] OASIS Cyber Threat Intelligence Technical Committee. Introduction to stix. https://oasis-open.github.io/cti-documentation/stix/intro, 2025.

[55] Tommaso Paladini, Lara Ferro, Mario Polino, Stefano Zanero, and Michele Carminati. You might have known it earlier: Analyzing the role of underground forums in threat intelligence. In *International Symposium on Research in Attacks, Intrusions and Defenses*, 2024.

[56] Panther Labs. Introducing the panther sigma rule converter. https://panther.com/blog/introducing-the-panther-sigma-rule-converter, 2024.

[57] James W Paulson, Giancarlo Succi, and Armin Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4), 2004.

[58] Pulsedive LLC. Pulsedive. https://pulsedive.com, 2025.

[59] Michael Sentonas. CrowdStrike Achieves 99% Detection Coverage in First-Ever MITRE ATT&CK Evaluations for Security Service Providers. https://www.crowdstrike.com/blog/crowdstrike-achieves-99-percent-detection-coverage-in-mitre-attack-evaluations-for-security-service-providers/, November 2022.

[60] Xiangmin Shen, Zhenyuan Li, Graham Burleigh, Lingzhi Wang, and Yan Chen. Decoding the MITRE Engenuity ATT&CK Enterprise Evaluation: An Analysis of EDR Performance in Real-World Environments. In *ACM Asia Conference on Computer and Communications Security*, 2024.

[61] Sigma. https://github.com/SigmaHQ/sigma/blob/master/Releases.md, 2025.

[62] Sigma. Main Sigma Rule Repository. https://github.com/SigmaHQ/sigma, 2025.

[63] SOC Prime. About us. https://socprime.com/about/, 2025.

[64] Splunk Inc. Splunk Security Content. https://github.com/splunk/security_content, 2025.

[65] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A Gerosa. Almost there: A study on quasi-contributors in open source software projects. In *International Conference on Software Engineering*, 2018.

[66] Klaas-Jan Stol and Brian Fitzgerald. Two's company, three's a crowd: a case study of crowdsourcing software development. In *International Conference on Software Engineering*, 2014.

[67] The MITRE Corporation. https://cve.mitre.org/, 2025.

[68] Rafael Uetz, Marco Herzog, Louis Hackländer, Simon Schwarz, and Martin Henze. You cannot escape me: Detecting evasions of {SIEM} rules in enterprise networks. In *USENIX Security Symposium*, 2024.

[69] Mathew Vermeer, Michel van Eeten, and Carlos Gañán. Ruling the rules: Quantifying the evolution of rulesets, alerts and incidents in network intrusion detection. In *ACM Asia Conference on Computer and Communications Security*, 2022.

[70] Apurva Virkud, Muhammad Adil Inam, Andy Riddle, Jason Liu, Gang Wang, and Adam Bates. How does Endpoint Detection use the MITRE ATT&CK Framework? In *USENIX Security Symposium*, 2024.

[71] Virus Total. Yara in a nutshell. https://virustotal.github.io/yara/, 2025.

[72] VirusTotal. https://www.virustotal.com, 2025.

[73] Dominik Wermke, Jan H Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. "always contribute back": A qualitative study on security challenges of the open source supply chain. In *IEEE Symposium on Security and Privacy*, 2023.

[74] Dominik Wermke, Noah Wöhler, Jan H Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. Committed to trust: A qualitative study on security & trust in open source software projects. In *IEEE Symposium on Security and Privacy*, 2022.

[75] Mingyi Zhao, Jens Grossklags, and Peng Liu. An empirical study of web vulnerability discovery ecosystems. In *ACM Conference on Computer and Communications Security*, 2015.

Metadata Field	Explanation
Title	short title describing rule
Status	rule stability (experimental, test, stable)
Description	longer description of rule `detection` logic
References	URLs to external resources contextualizing rule
Tags	maps to external frameworks (e.g., MITRE ATT&CK)
Author	rule author
Date	creation date of rule
Log Source	which logs to apply the `detection` to
Fields	fields in the log source to investigate further
False Positives	description of potential FP scenarios
Level	criticality (informational, low, medium, high, critical)

Table 5: **Rule Metadata:** Metadata attributes that are associated with Sigma rules.

A Differences Betweeen Sigma and Commercial Systems

In this appendix, we present examples of rules from Elastic and Splunk that diverge from their equivalent in Sigma.

A Case of Convergence and Divergence This is the most recent version of the rule represented in Figure 11.

```
`sysmon`
EventCode=10
TargetImage=*\\lsass.exe
granted_access IN (
"0x01000", "0x1010", "0x1038", "0x40", "0x1400", "0
   x1fffff", "0x1410", "0x143a", "0x1438", "0x1000")
CallTrace IN (
"*dbgcore.dll*", "*dbghelp.dll*", "*ntdll.dll*",
"*kernelbase.dll*", "*kernel32.dll*")
NOT SourceUser IN (
"NT AUTHORITY\\SYSTEM", "NT AUTHORITY\\NETWORK SERVICE
   ")
```

Credential Dumping with lsass.exe: Splunk

```
selection:
CallTrace|contains:
- dbgcore.dll
- dbghelp.dll
- kernel32.dll
- kernelbase.dll
- ntdll.dll
GrantedAccess|contains:
- '0x1038'
- '0x1438'
- '0x143a'
- '0x1fffff'
TargetImage|endswith: \lsass.exe
filter_main_system_user:
SourceUser|contains:
- AUTHORI
- AUTORI
filter_optional_thor:
CallTrace|contains|all:
- :\Windows\Temp\asgard2-agent\
- \thor\thor64.exe+
- '|UNKNOWN('
GrantedAccess: '0x103800'
filter_optional_sysmon:
SourceImage|endswith: :\Windows\Sysmon64.exe
condition: selection and not 1 of filter_main_* and not
    1 of filter_optional_*
```

Credential Dumping with lsass.exe: Sigma

The Impact of Crowdsourcing As discussed in the text, this case demonstrates that crowdsourced detection logic in the Sigma version of this rule is not integrated back in to the commercial vendor's version.

```
(
Processes.process_name=vssadmin.exe OR
   Processes.process_name=wmic.exe
)
Processes.process=*delete*
Processes.process=*shadow*
```

Deleting Shadow Copies: Splunk

```
selection1_img:
- Image|endswith:
- \powershell.exe
- \pwsh.exe
- \wmic.exe
- \vssadmin.exe
- \diskshadow.exe
- OriginalFileName:
- PowerShell.EXE
- pwsh.dll
- wmic.exe
- VSSADMIN.EXE
- diskshadow.exe
selection1_cli:
CommandLine|contains|all:
- shadow
- delete
selection2_img:
- Image|endswith: \wbadmin.exe
- OriginalFileName: WBADMIN.EXE
selection2_cli:
CommandLine|contains|all:
- delete
- catalog
- quiet
selection3_img:
- Image|endswith: \vssadmin.exe
- OriginalFileName: VSSADMIN.EXE
selection3_cli:
CommandLine|contains:
- unbounded
- /MaxSize=
CommandLine|contains|all:
- resize
- shadowstorage
condition: (all of selection1*) or (all of selection2*)
    or (all of selection3*)
```

Deleting Shadow Copies: Sigma

Production Versus Deprecation In this example, also discussed in the main text, Sigma maintains a rule that Elastic has deprecated.

```
selection:
AuthenticationPackageName: Kerberos
EventID: 4624
IpAddress: 127.0.0.1
LogonType: 3
TargetUserSid|endswith: '-500'
TargetUserSid|startswith: S-1-5-21-
filter_main_ip_null:
IpPort: '0'
condition: selection and not 1 of filter_main_*
```

Potential Privilege Escalation via Kerberos: Sigma

Example of Passive Crowdsourcing. In this rule, an author from Nextron Systems integrates references from Elastic and another security company, Fortra, to create a related Sigma rule.

```
selection_atexec:
  CommandLine|contains|all:
  - cmd.exe
  - /C
  - Windows\Temp\
  - '&1'
  ParentCommandLine|contains:
  - svchost.exe -k netsvcs
  - taskeng.exe
selection_other:
  CommandLine|contains|all:
  - cmd.exe
  - /Q
  - /c
  - \\\\127.0.0.1\\
  - '&1'
  ParentImage|endswith:
  - \wmiprvse.exe
  - \mmc.exe
  - \explorer.exe
  - \services.exe
condition: 1 of selection_*
```

Potential Impacket Lateral Movement: Sigma

Example of Selection Addition. Here, the Sigma version of a rule detects an additional commandline argument used with `bitsadmin.exe` when compared to the Splunk version.

```
(
  Processes.process_name=bitsadmin.exe OR
  Processes.original_file_name=bitsadmin.exe
)
 Processes.process IN ("*transfer*", "*addfile*")
```

File Download Via Bitsadmin: Splunk

```
selection_img:
  - Image|endswith: '\bitsadmin.exe'
  - OriginalFileName: 'bitsadmin.exe'
selection_cmd:
  CommandLine|contains: ' /transfer '
selection_cli_1:
  CommandLine|contains:
  - ' /create '
  - ' /addfile '
selection_cli_2:
  CommandLine|contains: 'http'
condition: selection_img and (selection_cmd or all of
   selection_cli_*)
```

File Download Via Bitsadmin: Sigma