

How It's Made: Uncovering Detection Engineering Processes for Network Intrusion Detection Rules

Koen T. W. Teuwen
Eindhoven University of Technology
Eindhoven, The Netherlands
k.t.w.teuwen@tue.nl

Emmanuele Zambon
Eindhoven University of Technology
Eindhoven, The Netherlands
e.zambon@tue.nl

Luca Allodi
Eindhoven University of Technology
Eindhoven, The Netherlands
l.allodi@tue.nl

Abstract

Many Security Operations Centers rely on signature-based Network Intrusion Detection Systems like Suricata, yet detection rule engineering remains understudied. We investigate this process by introducing SuriCap, a platform for rule engineering exercises, and hosting CTF-style workshops where 60 participants, trained MSc students, and experienced SOC professionals, created rules for four scenarios. Participants produced 3146 valid rules, enabling analysis of their methods, performance, and iteration patterns. Surprisingly, prior experience had limited impact on rule quality, suggesting that less experienced engineers can produce rules comparable to experts. We also observed challenges in generalizing rules beyond available tests, underscoring the need for sufficient labeled data. From our study, we identify three phases and a common pattern in rule engineering, offering SOC managers insights to improve their processes and expectations of engineer expertise.

CCS Concepts

• **Security and privacy** → **Usability in security and privacy**; **Intrusion detection systems**; *Network security*.

Keywords

Security Operations Center (SOC), Suricata, Network Intrusion Detection Rules, Detection Engineering

ACM Reference Format:

Koen T. W. Teuwen, Emmanuele Zambon, and Luca Allodi. 2026. How It's Made: Uncovering Detection Engineering Processes for Network Intrusion Detection Rules. In *11th Workshop on Traffic Measurements for Cybersecurity (WTMC '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3846375.3849098>

1 Introduction

Organizations constantly face security threats and take security measures to mitigate their effects. Among these measures, Intrusion Detection Systems (IDSs) such as Suricata [22, 38, 40, 41] are commonly deployed to monitor network traffic traversing relevant network links. Modern Security Operations Centers (SOCs) face challenges at both organizational and technical levels [17, 42]. Among these, the ingestion of new rules and updating of old ones is a key concern as assessing rule quality and relevance is hard, and

testing is expensive or impossible. Partially addressing these problems, some SOCs engineer their own in-house detection rules [40]. Widely available commercially engineered rulesets are sold as a product [8, 28], and open community rulesets exist as well [27], both of which can be used in conjunction with in-house rulesets to increase coverage [40]. Although industrial rule engineering processes lack extensive documentation, limited available public evidence suggests that rules are usually engineered given concrete samples of malicious and benign behavior [21]. Previous work has already investigated high False Positive (FP) rates within SOCs [2], in relation to rulesets [41], and to rules [38]. Although rule performance primarily depends on how a rule is developed, the current literature does not explore how rule engineers initially develop rules and how the adopted engineering strategies impact rule performance. Similarly, no study reports on the role of experience in rule quality or whether experts engineer rules differently from novices. Recent work covers the design aspects of detection rules [38], but to our knowledge, no studies evaluate the rule creation process.

We address these gaps by studying how network intrusion detection rules are engineered from incidents. Inspired by work in related fields [3, 6, 18], we organize several workshops akin to Capture the Flag (CTF) competitions in which participants engineer Suricata rules. To provide a realistic setup, we collaborate with a commercial SOC that provides expertise and training material used for their own analysts. Study subjects are recruited from a cohort of students at different levels, BSc computer science students with limited security training and MSc students with formal security training including intrusion detection, as well as expert engineers employed at the collaborating SOC. Participants access SURICAP, a platform developed for this study, where they engineer rules for different attack scenarios, submit them to receive feedback on correctness and performance, and iteratively refine them. We collect data on each participant's progress, yielding a rich dataset describing the engineering process, how rules evolve, and what strategies participants employ. Concretely, this study answers the following research questions:

- RQ1 How are network intrusion detection rules engineered?
- RQ2 How does the experience of rule engineers affect the engineering and performance of the developed rules?

2 Background and Related Work

Security Operations & Rule-based Intrusion Detection. A Security Operations Center (SOC) is tasked with monitoring to detect and resolve security breaches [42]. SOCs utilize network or host monitoring tools to collect data on assets, employing *Host-based Intrusion Detection Systems (HIDS)* to detect malicious activity on endpoints, and *Network-based Intrusion Detection Systems (NIDS)*, the focus of this study, to detect malicious network activity to/from



This work is licensed under a Creative Commons Attribution 4.0 International License. *WTMC '26, The Hague, Netherlands*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3017-7/2026/11
<https://doi.org/10.1145/3846375.3849098>

these devices. Suricata [23], which is the focus of this work, is the de facto standard NIDS and uses *rulesets* containing *signatures* for known threats. Snort [7] is another rule-based NIDS that uses a highly similar rule syntax for historical reasons. Each Suricata rule begins with a *header* specifying the *action*, *protocol* and *direction* of inspected packets, followed by a *body* containing additional options. Crucially, *detection options* specify which buffer is matched with which string, bytes, or regular expression. Some buffers support inspection after decompression or decoding, without requiring manual field location. *Content modifiers* change how content is matched (e.g., case sensitivity). Rules may combine positive and negative matches or specify a *distance* between them. The *flow* keyword restricts inspection to packets sent by servers or clients, while keywords like *flowbits* and *xbits* enable stateful detection across packets or connections. Rules can also use *threshold*, e.g., to control triggering frequency, and non-detection keywords like *metadata* to aid alert interpretation. Rule engineers may use arbitrary combinations of these features, where keyword order matters only for some detection options. Rules are used to detect malicious behavior via, e.g., Indicators of Compromise, bytes indicative of known exploits, or protocol deviations [26].

Prior literature studying rule(set) evolution [41] suggests a minority of rules causes the majority of alerts, and that most rules receive updates only to metadata rather than detection options. Since most rules never receive updates, the initial engineering process is critical, motivating our focus on this process before rules are first released. Related research on SOC workflows [40] found that many SOC teams dedicate significant resources to engineering in-house rules tailored to client environments. Unlike in-house rulesets, open and commercial rulesets are not designed to operate in a specific environment. Open rulesets [27] may include rules submitted by engineers with varying experience [35]. SOC teams commonly tune external rulesets by disabling rules [40] while re-engineering loops from reported FPs are uncommon, making the initial rule engineering process all the more influential on rule performance. Other work describes a SOC analyst attributing FPs to poor rule engineering [33]. The Suricata community [14, 34] has developed quality guidelines focusing primarily on syntax and structure. A recent study proposed six design principles to increase rule specificity while maintaining coverage [38], addressing features like alert throttling, matching content and method generalizability. However, these works address what rules should look like rather than how to derive such rules or how to apply the principles when engineering new rules.

Despite the recognized importance of detection engineering [24, 39], little is known about the initial rule engineering process. Prior work suggests using unit tests with positive and negative cases, backtesting on historical benign data, and staged deployment to prevent FPs [21]. To the best of our knowledge, our work presents the first academic study of the rule engineering process.

CTF-style Activities for Research. Research methods similar to ours have previously been applied to study reverse engineering and malware classification, where Jeopardy-style workshops are used to study how people address these tasks. One such work studies ‘perfect decompilation’ whereby participants work towards a predefined solution (i.e., pre-compilation source code) [6]. Another study on reverse engineering investigates the effects of experience on

strategies and performance [18]. Unlike [6], little feedback is given to participants and the platform uses a restricted focus viewer technique, blurring parts of the code to measure attention, across three challenges. A further related work studied how people classify malware compared to machines [3], investigating which features people inspect to derive conclusions about file maliciousness. Although the related work differs from our detection engineering domain (given the lack of a perfect solution), we draw inspiration from these studies, particularly the use of frequent submissions to better capture the overall process. Stimulating this, we incorporate syntax highlighting, clear feedback on errors, and a ‘baby’ challenge to help users get acquainted with the platform. Differentiating SURICAP and uniquely relevant to intrusion detection is its ability to assess the generalizability of submissions beyond the tests visible to participants.

3 Methodology

To answer the RQs presented in Section 1, we devise a methodology, summarized in Figure 1, supported by a dedicated measurement platform that allows us to capture data at-scale. The platform is designed to be non-intrusive, allowing participants to exhibit natural behavior, rather than forcing specific actions. Inspired by CTFs in related fields [6], we organize the activity as a CTF where participants are presented with various challenges in the form of *scenarios* and compete with each other within the time of the activity for the best coverage and specificity of their engineered rules. Participants tackle scenarios by inspecting network traffic packet captures (PCAPs) and engineering Suricata rules to detect those scenarios. During this process, participants receive feedback on their submissions from a limited number of *visible tests* which they can use to refine their rules. Additional *hidden tests* are performed to reflect real-world scenarios [21]. We include both positive (malicious) and negative (benign) (in)visible tests to assess coverage and specificity, which are commonly regarded the most important performance dimensions of detection rules [38, 40, 41]. This mimics the unit and backtesting [21] described in Section 2 whereby rules are initially engineered given a positive and a negative test, and then further tested on unseen data in staging and production environments.

Participants physically attend the two hour workshop to address challenges, while a leaderboard displays current scores on the visible tests. To make the workshops educational whilst not biasing experimental outcomes, participants can, at any time during the workshop, ask the organizer of the CTF to clarify objectives or for assistance with the Suricata rule language, but will receive no directions for solutions to challenges. Before workshops, participants receive personalized instruction through a video lecture and a handout.

The experiment is run using a gamified platform, SURICAP to collect data on rules while they are being engineered enabling research on these processes from start to end (RQ1). From the participants’ point of view, SURICAP enables competition whilst easing rule development through automated feedback. The workshop is freely and voluntarily accessible to everyone. We advertise the workshop in cybersecurity courses delivered by European technical universities. To access the workshop, interested participants complete an intake questionnaire to gauge relevant background and experience (RQ2).

We randomly divide participants in two groups to evaluate if knowledge of rule design principles [38] impacts rule engineering

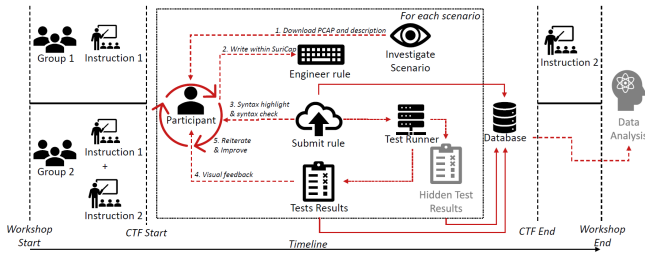


Figure 1: Experiment design to support answering of RQs.

(outcomes). A randomly defined treatment group receives an additional instruction on the design principles prior to the activity, whereas the control group only receives the standard SOC training.

During the activity, participants are asked to engineer Suricata rules to detect malicious traffic by raising a single alert while remaining insensitive to benign traffic. Participants know the attack technique (e.g., Command & Control over encrypted channel) contained in each scenario, but are not given any specifics on attack triggers or characteristics they could use for detection. Participants can inspect a PCAP to determine suitable detection characteristics for engineering rules. Participants can submit the rules they are developing to the backend system to receive automated feedback in the form of success/failure to (not) detect malicious/benign traffic in the visible tests, which they can use to further refine their rules. Additional information such as error messages or suricata CLI output are also provided to the user through the interface. Participants could submit rules as often as desired. The hidden tests are run to measure the coverage and specificity beyond the PCAPs visible to the participants. This replicates real-life circumstances of the rules being deployed against data they were not tested against. To reproduce this in full, the visible test tests rules against the oldest samples, whereas the older, often more complex malicious behavior (e.g., using evasion techniques) is tested against by the hidden test. Each test has associated Suricata configuration options to ensure variables that can be used in rules (e.g., \$HOME_NET) are defined appropriately for the sensor setup and scenario represented by the test. Hereafter, we first describe ethical considerations, how we recruit participants and the process they undergo for the experiment. Subsequently, we detail scenarios for which we measure the rule engineering process and elaborate on the measurement platform.

Ethics considerations. This research was carried out with approval from our institution’s ethical review board under the approval number ERB2024MCS34. We obtained explicit informed consent from all subjects whose data are used in this study and assured that the research would not affect their study/work conditions. Participation in the activity is non-obligatory and solely encouraged through the educational and entertaining aspects. Moreover, participants of the activity are not obliged to participate in the research. The experiment design, including the presence of hidden tests, is explained prior to the activity, and sample solutions are provided afterward. Measurement of user input occurs non-intrusively through explicit user input. After data collection and correlation, personal information is removed so that researchers cannot use the data to identify individuals to whom data relates.

Subjects Recruitment. We recruit participants through academics at our own and other institutions, who helped us organize the workshop locally and announce the workshop to potentially interested students. We also reached out to student communities (i.e., CTF associations) at those universities to further promote the workshop. An announcement was also shared within their respective communities. Although we aimed primarily at recruiting MSc students following a cybersecurity program, the workshop was explicitly open to anyone, resulting in the recruitment by word of mouth of also the BSc students and PhD graduates. The recruited participants fit the job profile for junior SOC roles [16] and are similar to rule engineers who occasionally contribute to open rulesets. This led to the organization of five workshops at four institutions¹, in which a total of 60 people with a wide variety of backgrounds participated. 85% had previously completed a BSc degree or equivalent as highest education level. 62% of the participants learned about the activity through a cybersecurity MSc course they attended, whereas 30% learnt about the activity through a group of CTF enthusiasts. These statistics suggest that the majority of all participants are indeed currently following a MSc program with a focus on cybersecurity. 80% of the participants identify as male, whereas 17% identify as female.

Further, three experts from the collaborating SOC accepted to join the experiment.² The experts all have years of experience in incident detection and detection engineering using Suricata. One expert has 6 years of experience and is currently a senior analyst at the collaborating SOC. Another expert has 3 years of SOC experience related to triage and infrastructure. The third expert has 5 years of experience running the SOC’s detection infrastructure.

Preparatory Instructions. One week before the experiment, participants receive a preparatory video lecture. The video lecture is based on the training material provided by the collaborating SOC for the training of their own analysts, and includes information on the experiment design, required tools, and the platform the participants will interact with. The treatment group receives an additional instruction on the design principles from [38]. The instruction (Appendix A) familiarizes participants with relevant prerequisite knowledge on the tools they will be working with, including Suricata and the platform. All participants receive a handout with the material covered during the video lecture, which is also made available on the platform during the exercise. The usage of a video lecture rather than an in-person lecture ensures that all participant in the control/treatment group receive identical instructions.

User questionnaire. In order to control for possible confounding factors in our subject pool we design a questionnaire to gauge relevant prior experiences of participants that may affect rule engineering processes or experimental outcomes. The questionnaire (Appendix B) covers network protocols (e.g., TCP, TLS), and offensive computer security, to the extent that these relate to the scenarios presented in Section 3. These topics have previously been identified as relevant for Tier 1 SOC analysts [29]. The questionnaire also covers Wireshark, intrusion detection, and Suricata, which are the core technologies participants are expected to utilize during the

¹Extraordinary circumstances resulted in the moving of one workshop, which negatively affected participant turnout, which prompted us to organize a second workshop at the same institution.

²While recruitment of senior analysts is scarce due to limited availability, the experts serve as a meaningful separate comparison benchmark against other participants.

activity. The questions about offensive computer security specifically relate to the tactics of the scenarios discussed hereafter. We devise questions to measure experience with relevant topics to avoid reliance on self-assessment. For example, instead of asking participants directly how experienced they are in working with network protocols, their experience is gauged using questions like: “*how does TCP ensure delivery of packets in the correct order?*” We also collect demographics like age and education level to describe the population and inquire how frequently participants attend CTFs. A control question was included to assess respondent reliability.

Scenarios design. To ensure our analysis covers a wide range of NIDS rules, the experiment consists of one “baby”, and three main scenarios, each focusing on one common MITRE ATT&CK tactic [9] that can be detected by network monitoring. Furthermore, the scenarios cover rules for which detection rules are commonly engineered.³ Concretely, the scenarios focus on Reconnaissance (i.e., scanning for unintended information disclosure), Initial Access (i.e., payload delivery through a download following malspam), and Command and Control (i.e., C2 communications from a RAT). These are chosen to be representative of tactics commonly detected by NIDSs [38]; further, similarly to [18], we must find a balance between the time we expect participants to invest in the activity and the data we can collect. Scenarios are designed to allow for both simple and complex solutions, which although may both pass the visible tests and appear successful from a participant’s point of view, may perform differently on the hidden tests. In designing the tests for scenarios, the working principle is that visible test data is chosen that would likely be available to a rule engineer when implementing a rule, which means it is limited [21]. Following this principle, the oldest network traffic, or traffic concerning the most prevalent system type are visible to participants whereas more recent data or traffic corresponding to more rare systems are hidden from participants. This ensures the realism of the data availability at the time of rule engineering, as the ‘training process’ (i.e., the rule engineering) cannot rely on data from the future [4].

All participants start with a ‘Baby’ warm-up scenario, followed by the three main scenarios in randomized order. Once a user has obtained the maximum possible score on the visible tests the next scenario is unlocked. To prevent circumstances in which a user may be deterred because they fail to obtain this score, the next scenario is unlocked after a user spent 5 minutes working on that scenario.⁴ The user is free to go back to any previously unlocked scenario anytime. Hereafter, we exemplify how the *Information Disclosure* scenario and the benign tests (shared among all scenarios) are derived. The derivation of the other scenarios is presented in Appendix C. *Information Disclosure.* The *Information Disclosure* scenario covers an unintended information leak detailing web server configurations through the built-in `phpinfo` function, from which adversaries can derive installed software, as well as server configuration settings. A SOC may choose to detect this type of behavior as precursor of follow-up scans or attacks, or for forensic reasons during incident reconstruction⁵. In collaboration with SOC experts, we collect

```

alert http $HOME_NET any -> $EXTERNAL_NET any (
  msg:"Good phpinfo rule"; flow:established,to_client;
  file.data; content:"<title>"; content:"phpinfo"; within:20;
  threshold:type both, track by_src,count 1, seconds 180;)

alert http $EXTERNAL_NET any -> $HOME_NET any (
  msg:"Suboptimal phpinfo rule";
  http.request_line; content:"/phpinfo."; flow:to_server;)

```

Figure 2: ‘Good’ (top) and ‘Suboptimal’ (bottom) rule detecting Information Disclosure

PCAPs from servers with various PHP versions where disclosure was successful, with and without the use of evasive methods, like applying base64 encoding to the URL [36]. Similarly, we collect PCAPs from servers where information disclosure failed to test for FPs. We also collect PCAPs of access to websites with similar content but without information disclosure, such as pages documenting the `phpinfo` function, which could trigger FPs, in case the rules lack specificity in their match definitions. Participants see the performance of the rules they submit evaluated only against the capture corresponding to a successful disclosure against the server running the oldest PHP version in line with our working principle. The remaining 23 PCAPs, which represent more recent or more complex instances of the same behavior, are used as hidden tests.

To exemplify how the various tests behave with respect to different rules, we explain how two examples (see Figure 2) may fail to trigger true positives or trigger FPs. The ‘good’ example obtained a perfect F1-score of 1.0 across the all tests combined whereas the ‘suboptimal’ rule obtained an F1-score of 0.56. The primary difference is that the good rule detects server responses instead of client requests such that only successful attempts are detected, in line with the design principles from [38]. Instead, this rule also triggers on several tests corresponding to unsuccessful attempts, thus resulting in FPs. Moreover, the suboptimal rule only captures one of the possible ways in which the `phpinfo` function may be invoked, thus missing cases in which the URL was encoded, or the `phpinfo` function was invoked from web resources not named “phpinfo”.

Benign Traffic. We include several PCAPs that contain only benign traffic from [12]. The benign PCAPs are chosen to include a variety of HTTP traffic to test for FPs from any rules that may aim at detecting the HTTP traffic from the *Information Disclosure* scenario and *Malspam Download* scenarios. Similarly, benign PCAPs that contain TLS traffic are beneficial in evaluating rules for the RAT scenario. These PCAPs contain no malicious traffic, and high-quality rules are not expected to raise any FPs on them. In addition to the scenario PCAPs selected to be made available to participants, we also include a capture containing benign traffic originating from a Windows 7 PC. This capture contains a variety of benign behaviors, like usage of a web browser, cloud storage providers to download files, and internet radio, along with traffic generated by background services running on the device. In line with our working principle, we consider this capture to be representative of (older) common traffic, since Windows is the prevalent desktop operating system [32].

Three additional more recent benign PCAPs are selected as hidden tests to evaluate whether rules generalize to unseen benign traffic without raising FPs. Another more recent capture from a

³63% of all rules from the ETOPEN ruleset focus on the HTTP and TLS protocols.

⁴This proved to be mostly irrelevant: in 89% of cases participants only moved to the next scenario after passing the visible tests on their current scenario; conversely, participants who moved on to the next scenario without passing these tests spent on average 22 minutes on that scenario.

⁵For example, see ET rule 2019526.

Windows 7 PC containing similar traffic, but also video streaming, is leveraged as an addition to the older capture to evaluate whether engineered rules may raise more FPs on traffic collected after the traffic for which the rule was originally engineered. To assess whether rules may trigger FPs on other Operating Systems and/or previously unseen protocols, we select a capture containing normal traffic from a Debian host performing various normal actions including P2P Deluge downloads, web browsing, video streaming, and using Jabber. Furthermore, to assess the potential for FPs when interacting with various web servers that potentially run different software with different configurations, we select a benign PCAP that contains interactions with Alexa top 1000 domains.

The SURICAP Platform. To execute the methodology, a measurement setup is required to observe and collect data on the rule engineering process. To this end, we propose SURICAP: a measurement platform to study and evaluate Suricata rules. SURICAP acts as a wrapper for Suricata: it captures user input in the form of rules and automatically provides feedback to the user while hiding Suricata's CLI and configuration. Figure 3 provides an overview of the SURICAP interface. The menu bar at the top provides easy access to the instruction handout and the Suricata documentation [23]. On the top left is provided a description of the current scenario. Each scenario has at least one associated PCAP containing malicious behavior that the rule developed by the participants should detect. Scenarios can contain additional instructions in the form of a textual description or a file (i.e., PDF) to provide relevant information to users, such as the context from which the traffic was captured. All scenarios described in Section 3 contain a brief overview, without suggesting specific characteristics for detection. Users can download the network traffic from the User Interface (UI) to inspect the PCAPs using Wireshark. When a user *checks* a rule written in the upper-left input box on the left, feedback in the form of syntax highlights in the upper right and Suricata output in the upper right indicating any potential syntactical errors. When a user *submits*, the lower input boxes on the left will display the most recently submitted rule, and tests will be run in the back-end to render test results in the far bottom right in addition to the feedback generated by a check. The tests and results thereon are shown on the right, including the expected test outcome and PCAPs.

Similarly to prior work [6], SURICAP provides swift feedback whenever a rule is submitted to incentivize frequent rule submission and highlights syntax by formatting rules, splitting options to separate lines and coloring keywords and special characters. The platform also indicates whether Suricata was able to successfully load the rule, showing the user detailed Suricata output to correct invalid rules. Users receive color-coded (i.e., green for successes, red for failures) feedback indicating whether the malicious behavior is correctly detected, and whether benign data triggered FPs. Additional details regarding the number and nature (i.e., Suricata Fast and EVE logs) of FPs can also be accessed through the UI.

To reinforce the gamification element and similarly to [6], during the workshop participants are awarded a score based on the tests to compete with other participants on a leaderboard. The score of participants is computed using their best submission instead of their most recent submission. In case multiple participants obtain the same high-score, the participant who first obtained that score will

win. Participants are informed that at the end of the workshop the 'hidden leaderboard' based on all tests including the hidden tests will be revealed to encourage them to continue working on their detection rules after they have passed all visible tests to enhance rule generalizability. The winners receive no awards beyond the fame resulting from the announcement to encourage intrinsically motivated participants. Through the provision of these various types of feedback and the inclusion of time in the scoring method, we stimulate frequent submissions to establish a detailed dataset.

4 Data Analysis

We organized 5 CTF sessions to collect data from 60 participants with varying experience levels (RQ2) who jointly submitted 3146 valid rules. From these rules, we consolidate 5176 individual changes (e.g., adding or modifying a Suricata option) that describe engineering processes followed by participants (RQ1). Combined with the 15751 test results for all valid rules (i.e., rules parsable by Suricata), we evaluate the engineering process that analysts follow to develop their rules and experience effects in relation to rule quality.

Experiment progression. Table 1 presents an overview of the data collected on rule engineering processes for each scenario and across all combined scenarios. We can see that most participants obtained good F1-scores on the Baby scenario and passed the visible tests on average after 7 minutes. Some participants had to get acquainted with the Suricata syntax at the beginning of the activity, as depicted by the lower fraction of valid submitted rules for the Baby scenario. Note that many participants failed to reach optimal F1-scores for the main scenarios and there is a high variance therein. The recall scores, which are lower than the precision scores, suggest that participants struggle more with coverage than specificity. These results suggest that different rules result in different specificity/coverage and that many rules could possibly be improved. Furthermore, the data suggests that participants struggle the most with engineering performant rules for the Malspam Download scenario, as indicated by the lower F1-scores, more time required to pass the visible tests, and the increased rule submissions. Note that recall scores obtained by participants (0.59) are only marginally lower than those obtained by the expert group (0.65) or relevant public rules⁶.

Figure 4 shows the participants' progression on the main scenarios, with the x-axis representing the fraction of time spent towards the final submission. The top row shows how many participants have submitted a valid rule for that scenario. Note that many participants are initially not implementing a rule but instead investigating the behavior they should detect and considering how to engineer the rule(s). We identify this as **Phase 1** of the rule engineering process. We observe a sharp increase in participants who have submitted a valid rule near the end, suggesting these participants spent significant time thinking before implementing a rule. Note that the distributions in the bottom two rows are computed over increasingly many samples over time as more participants have submitted a valid rule. When inspecting the bottom two rows, it becomes evident that participants achieve good specificity early on as suggested by the high precision scores⁷, whereas coverage is

⁶ET rules 2035607 and 2035595 obtain recall of 0.67 on the RAT scenario and ET rule 2019324 obtains a recall of only 0.25 on the Malspam Download scenario.

⁷In the case where a rule triggers zero alerts, a precision score of 1 is assigned.

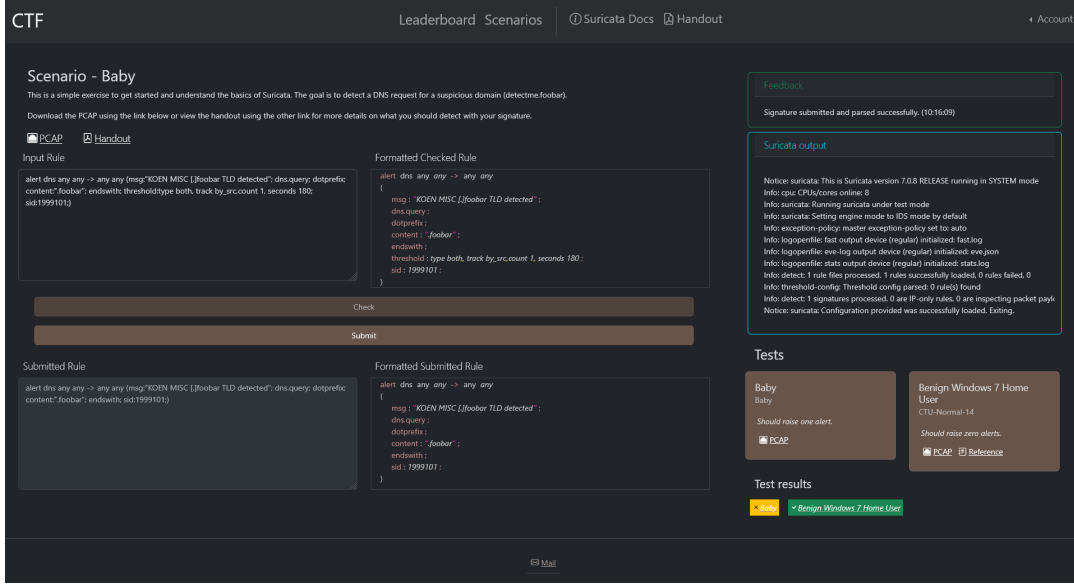


Figure 3: A screenshot of the UI of SURICAP, showing the menu bar containing resources such as the documentation and the handout, the scenario description, original and syntax highlighted user inputs, and test descriptions and results.

Table 1: Statistics describing the engineering process of the average (mean and SD) participant.

Scenario	F1-score	Precision	Recall	Number of Design Principle Issues	Time (scnds) to Pass Visible Tests	Number of Submitted Rules	Fraction of Valid Submitted Rules
Baby	0.98 ± 0.13	1 ± 0	0.98 ± 0.13	3 ± 1	430 ± 592	22 ± 18	0.66 ± 0.19
Information Disclosure	0.59 ± 0.23	0.88 ± 0.16	0.51 ± 0.25	3 ± 1	667 ± 685	17 ± 18	0.73 ± 0.22
Malspam Download	0.48 ± 0.26	0.96 ± 0.16	0.37 ± 0.27	3 ± 1	1273 ± 972	22 ± 25	0.74 ± 0.22
RAT	0.60 ± 0.28	0.94 ± 0.20	0.52 ± 0.27	3 ± 1	968 ± 689	19 ± 17	0.78 ± 0.18
All scenarios	0.67 ± 0.15	0.84 ± 0.16	0.59 ± 0.19	10 ± 4	2931 ± 1336	75 ± 48	0.66 ± 0.19

often lacking until the end of their progression as suggested by the low recall scores that only increase near the end. Coverage is also the characteristic of rules that participants end up improving the most as suggested by the trends. We identify the implementation of a rule that detects the relevant behavior as **Phase 2** of the rule engineering process. We manually inspected several rules with simultaneous low coverage (recall 0.0) and high specificity (precision 1.0), to determine potential explanations for this peculiar performance. Some rules specify the wrong traffic direction in the rule header (i.e., IP address groups or port numbers). Others contain *pcr* regular expressions that contain minor bugs causing them to not match any relevant traffic. Other rules have appropriately defined a characteristic to detect, but failed to specify where or how data to be matched should be preprocessed (i.e., not applying keywords like *file.data* to prepare an HTTP body for processing).

Progression beyond passing visible tests. During the CTFs and while speaking with participants, we observe that participants struggle to improve the performance of rules on the hidden tests. We identify improving a rule beyond the available test data as **Phase 3**

in the rule engineering process. Interestingly, this observation relates to the impact of availability of test data (i.e., access to hidden tests) on the rule engineering process. Verifying this observation using the data, we zoom in on the process participants follow after passing the visible tests (i.e., obtaining an F1-score of 1.0 thereon). Concretely, we compare the performance on the hidden tests after having passed the visible tests to that when the visible tests are first passed. Figure 5 shows resulting comparisons, where the different columns cover the three main scenarios. The top row indicates how many participants have passed the visible tests and submitted subsequent valid rules also passing these tests. The bottom row depicts the distribution of offsets in the F1-score on the hidden tests, i.e., the difference between the score of the most recent submission and that of the first submission to pass the visible tests.

When inspecting the top row of Figure 6 w.r.t. how many participants submitted a valid rule as in Figure 4, we observe that a significant number of participants never proceeds to this phase since they submit no new valid rules passing the same visible tests after initially passing them. The bottom row of Figure 5 suggests that most participants do not significantly or consistently improve

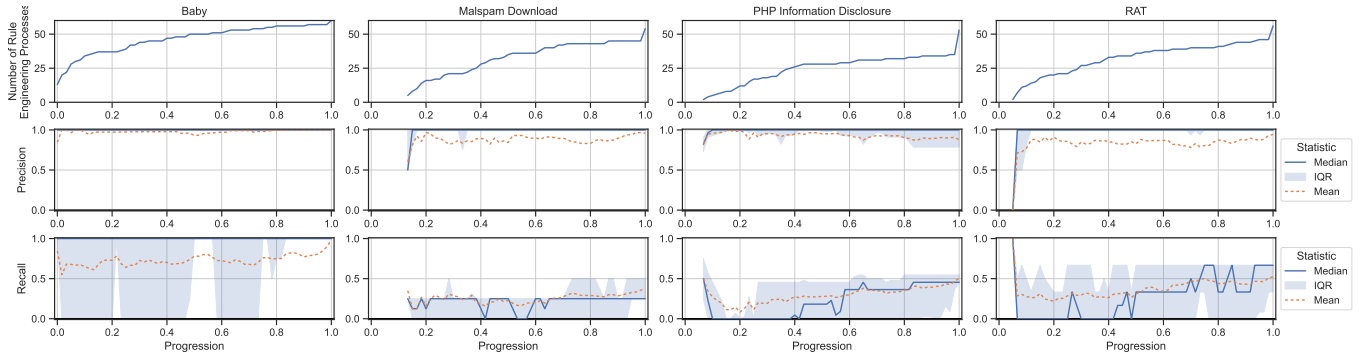


Figure 4: Progression of participants for each main scenario showing the number of participants submitted a valid rule (top), and the Precision (middle) and Recall (bottom) on all tests of their most recent submission.

rule performance (i.e., F1-score on all tests combined) beyond the already obtained performance. Attempts of participants to generalize rule performance without knowledge on hidden tests can harm performance as indicated by the sharp decline in mean F1-score offset for the Information Disclosure scenario, which later averages out closer to zero as more participants pass the visible tests.

Rule Engineering Phases. In Section 4 and Section 4 we identified three phases of the rule engineering process. The phases can be summarized as: **(1) plan, (2) perform, (3) perfect**, whereby rule engineers first plan the rule design, then perform the implementation, and finally perfect it to generalize. Figure 11 (Appendix E) shows how many participants are in a certain phase. The number of participants in Phase 1 initially remains constant and then quickly decreases, after which the decline stagnates. About half of the participants move on from Phase 1 within about 35% of the time they spent on a scenario in total, suggesting that on average 35% of their total time is spent thinking about how to design a signature for a scenario. The number of participants in between Phases 1 and 2 remains overall low, suggesting that few participants struggle with the Suricata syntax. The number of participants in Phase 2 gradually increases as people move on from Phase 1, and the number of participants in Phase 3 begins to increase shortly thereafter. Importantly, we note that only about half of all participants eventually progress to Phase 3. We observe that most participants remaining in Phase 2 submit no new rule after having passed the visible tests. Several possible explanations exist for this phenomenon. Some participants may choose to quickly move on to the next scenario once they have built a rule that passes to visible tests, to obtain higher (visible) scores faster for the CTF. Others may be unsure how to improve rules and therefore not initiate such attempts. Other participants may not have time to attempt improvements for all scenarios.

Subject background and the effect of rule design principles. To evaluate whether the background knowledge of a participant helps in engineering effective rules, we consider both their previous knowledge, as gauged with the questionnaire described in Section 3, and their knowledge of rule design principles introduced in [38] (i.e. whether they receive the training material). To assess the effect of experience on the rule engineering process, we regress over scores for various experience categories on the questionnaire to

predict three key performance metrics: (1) the F1-score, (2) the number of design principle issues detected by *suricata-check*[37], and (3) the time required to pass the visible tests. The first two metrics relate to the quality of the engineered rule and the third metric pertains to the cost associated with the rule engineering process. The regressions do not suggest prior experience has a significant effect on the performance of the 60 participants with respect to any of the three metrics. Therefore, experience appears to have negligible effects on performance of the rule engineering process and of resulting rules. The regressions, together with 95% confidence intervals, are presented in Appendix D.

To assess how knowledge of the rule design principles impacts performance, we compare the metrics between the group that received the related instruction material and the one that did not. Figure 6 shows the performance of the two groups. Subfigure (1) shows that the participants in the treatment group have similar F1-scores. According to a Mann-Whitney U test, the difference is non-significant. Subfigure (2) shows that participants in the treatment group engineer rules that generally, albeit marginally, adhere better to the design principles ($p = 0.04$) than those of users in the control group. Subfigure (3) shows that there are no clear statistical differences between the two groups in terms of time spent to first pass the visible tests. From these data, it emerges that the treatment and control groups only minimally differ in adherence to design principles, which does not lead to significant differences in the performance of the rules. However, our experiment suggests that the adoption of design principles may lead to a trade-off with increased engineering time although we did not observe a clear statistical difference. This suggests more research is needed on how to train rule engineers and to balance the trade-off between increase in performance and time/resources needed to engineer rules. Due to the similar performance between the two groups, we disregard the distinction between groups for the remainder of the paper.

Uncovering rule engineering strategies. Rule engineers follow different strategies to implement rules during Phase 2 and Phase 3. To identify different strategies for implementing rules, i.e., the order in which rule keywords are addressed during implementation, we use unsupervised learning to cluster similar strategies and then describe the clusters. First, rule implementation processes are modelled as variable-length sequences of categorical variables representing the

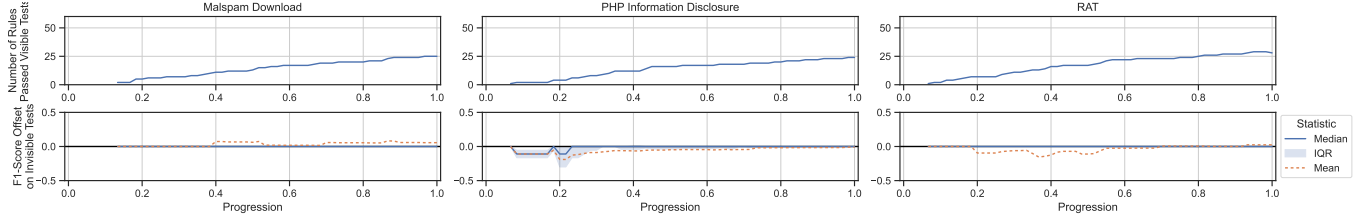


Figure 5: Progression on main scenarios after passing the hidden tests showing how many participants pass the visible tests (top) and the offset in F1-score on the hidden tests of their most recent rule w.r.t. their first rule passing the visible tests (bottom).

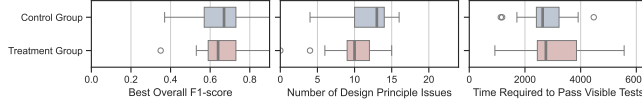


Figure 6: Comparison of the control and treatment groups.

steps taken in the process (i.e., addressed). Then, Sequence Graph Transform (SGT) [30] embeds these sequences into fixed-length numerical vectors. SGT leverages the probability that a categorical variable is followed by another categorical variable within some distance to encode the order of keywords addressed in each process. Finally, HDBSCAN [19] derives clusters, representing strategies, which we characterize to understand of how people engineer rules.

To model rule implementation processes as variable-length sequences of categorical variables, we automatically trigger whenever a participant checks or submits a new rule to detect which fields are added, modified, or removed since the previous check or submission. Many rule fields are protocol-specific and scenarios use different protocols. Thus, we cannot use the fields directly to synthesize process steps across scenarios. To allow across-scenario analysis, in collaboration with two SOC experts, we define a mapping from the low-level Suricata fields to a high-level *field type*. Field types either (1) specify the inspected *traffic direction* (e.g., `src_ip` in the header and flow), (2) control *payload matching* (e.g., content, and depth), (3) deal with *application layer* protocol aspects (e.g., `tls.cert_issuer` and `base64_decode`), (4) add *stateful* components (e.g., threshold and flowbits), or (5) describe *non-functional* aspects (e.g., `msg` and `fast_pattern`). To create the sequences, we simply concatenate the field types of all changes in chronological order⁸. This resulted in 223 sequences.

We use SGT to transform categorical variable-length sequences into 16-dimensional numerical length-sensitive embeddings capturing long-term interdependencies between categorical variables. We use HDBSCAN with Manhattan distance, which is suitable for high dimensionalities [1], to perform unsupervised clustering of embeddings and derive homogeneous clusters of engineering processes. This process⁹ results in the five clusters/strategies shown in Figure 12 (Section F). HDBSCAN rejected 152 engineering processes, suggesting that many processes cannot be grouped into a homogeneous cluster, and there are potentially many possible strategies. Cluster 4 is the largest cluster with 28 processes and represents the most common strategy, whereas Cluster 1 is the smallest

cluster with 7 processes. In Subfigure (1) of Figure 12, Cluster 4 appears isolated from most other processes. Subfigure (2) shows no strong concentration of processes by scenario, suggesting specific scenarios do not necessarily prescribe following a certain strategy. Only Clusters 1 and 2 do not contain processes from all scenarios.

Figure 7 provides a density distribution of the changes applied to the submitted rules by the expert group. Additionally, it shows how many such changes are made. Similar figures for all the other clusters discussed here are in Appendix F. One commonality in strategies between the expert group and most of the clusters pertains to the order of changes to field types. Most clusters including the expert group initially focus on fixing the traffic direction. Application-layer and payload-matching changes follow, with approaches varying across clusters. This appears most applicable to Cluster 4, but also to Clusters 2 and 3, albeit with a more uniform approach, whereby changes are applied throughout. Stateful aspects of rules (e.g., alert throttling) tend to be the last thing participants focus on for all clusters. A stark example is Cluster 4 wherein changes to stateful properties of the rule are solely made at the end.

Overall, the different clusters highlight very different strategies. Clusters 0 and 1 represent ‘one-shot’ rule engineering strategies, whereby all changes are applied mostly at the end. Cluster 2 adopts a U-shaped approach whereby most changes are applied at the start and end, suggesting a more analytical approach towards the middle of the engineering process and a fine-tuning at the end. Cluster 3 applies changes throughout in a uniform fashion, without clear spikes but with relatively many submissions. This suggests an engineering process focused on a continuous testing-improving cycle, whereby engineers continuously test new changes and make improvements as they apply them. Cluster 4 strikes a middle ground between Cluster 2 (U-shaped process) and Clusters 0/1 (most changes at the end). This suggests a relatively high prevalence of fine-tuning towards the end but with a more gradual rule engineering process.

Clusters also differ in terms of the rule features they implement. For example, Cluster 1 differs from Cluster 0 as it does not modify any stateful property of the rule. Cluster 2 differs from the other clusters in that no explicit payload matching keywords and no stateful keywords are used. Rules resulting from this strategy may instead rely on matching of IP addresses, which are specified as part of the traffic direction, which is the field type modified the most for Cluster 2. Cluster 3 does not modify any stateful keywords and contains more overall modifications than other clusters. Cluster 4 is most similar to the expert group, although the expert strategy makes modifications earlier, specifically for the stateful keywords.

⁸If an update contains multiple changes we also apply alphabetical order.

⁹Unless specified otherwise, default hyperparameters are used.

The expert group first focuses on non-functional and traffic direction keywords, closely followed by application layer keywords, after which they turn to payload matching and finally on stateful keywords. The order in which experts address various keyword groups appears to be better separated, although similar to Cluster 4. Specifically, the distinction in order between payload matching and application layer keywords is more explicit for the expert strategy. Note that experts consistently engineers rules with high F1-scores.

Figure 8 compares the effectiveness of the different strategies represented by the clusters. Most performance measures obtained from the various clusters of engineering processes are similar for most scenarios, and where dissimilar no one strategy consistently under- or outperforms. Subfigure (4) suggests that processes from Cluster 4 have relatively few submitted rules in all scenarios, possibly indicating a well-thought-out process. Overall, we find no clear difference in outcomes across strategies, suggesting that forcing (e.g., through training or technological aids) a specific process upon rule engineers may lack tangible benefits in outcome. This suggests that rule engineering is both an engineering and a creative process.

Fine-tuning behavior. We further characterize rule updates in rule engineering processes by examining the edit distance using two metrics. The Levenshtein distance, relative to the rule length prior to the update, quantifies the portion of characters that was changed in an update¹⁰. Secondly, multiset-Jaccard similarity [10] measures similarity between the rule design before and after the update in terms of the Suricata keywords used. Unlike the Levenshtein distance, the Jaccard similarity insensitive to the Suricata syntax.

Figure 9 depicts the joint distribution of the relative Levenshtein distance to the Jaccard similarity. The figure suggests that the most updates only marginally change the rule as observed through the low relative Levenshtein distance and do not alter the rule design as indicated by the peak frequency of Jaccard similarities at 1.0. These minimal rule updates reflect what we refer to as *fine-tuning behavior*, meaning rule engineers primarily gradually refine rules over time, rather than pivoting to entirely different designs.

5 Discussion

Although rule engineering is not trivial, as reflected by the low recall of detection rules (Section 4, the results of Section 4 suggest that experience has no significant effects on the quality of the rule or the efficiency of the process by which that rule was derived. This suggests that inexperienced rule engineers may be capable of effectively engineering detection rules of sufficient quality, following a basic instruction and even with minimal background.

However, our results also emphasize the importance of test data availability to rule engineering, as participants appear unable to generalize rules after passing the visible tests. In other words, it is difficult for rule engineers to develop signatures that generalize beyond the incidents and environments for which a rule was engineered. We find that increasing opportunities for rule engineers to test against more environments or incidents, or feedback loops as previously suggested [40] may promote rule quality and should be further explored in future research. Especially considering that

many rules never receive updates [41], the initial engineering process is crucial for rule quality. SOC's may benefit from employing methods to increase test data availability. For example, SOC's may integrate feedback loops to report FPs with test data to engineers to enhance the generalizability of rules. Increased automation for rule engineering, including the collection of and testing on data, and possibly automatic generation of rules, is another opportunity.

Since results from Section 4 suggest experience has no significant effect on rule engineering performance, rulesets established through community efforts as described in Section 2 may contain rules of similar quality to commercial rulesets if the experience of rule engineers is the only differentiating factor. Therefore, the idea of complementing internal rulesets with external (e.g., open) rulesets to increase coverage [40] may benefit SOC's. Future research may further explore such differences, specifically w.r.t. the test data availability. Although rule-level quality may be similar, ruleset-level quality may differ as a result of how the ruleset is managed.

We identified three rule engineering phases and a common approach followed in the implementation, which is shared with the expert group. The phases are discussed in Section 4 whereby rule engineers first plan the design of a rule, then perform the implementation, and finally perfect the rule. We found several commonalities between all implementation processes (i.e., phases 2 and 3), suggesting the existence of a common strategy. Rule engineers first focus on aspects surrounding detection characteristics, such as the circumstance in which a characteristic can be observed by implementing the direction of the network traffic. Subsequently, they move on to implementing the detection of the characteristic(s) itself using application layer and payload matching keywords. Cluster 4, which follows this approach and is the largest cluster, shows a high similarity to the expert group in terms of the strategy followed. SOC's may organize internal processes around these phases and the common strategy to streamline workflows. Moreover, we found that rule engineers focus updates on fine-tuning rules as opposed to pivoting to alternative rule designs. On the other hand, our results suggest that rule engineering is as also a creative process, following different strategies, focusing on different aspects of rule engineering, which appear to remain largely equivalent in terms of performance outcomes. This suggests that strict training or engineering processes, for example enforced through the use of supporting technology, may lead to negligible or no benefit in terms of rule quality and may boil down to solely a standardization effort. Rule engineering training and support could therefore focus on rule completeness rather than the engineering processes, for instance, by ensuring all relevant aspects like design principles are considered. Further research is needed in this direction.

Limitations. Due to the high similarity between Suricata and Snort, we expect our findings to be directly applicable to Snort rules. Although Suricata and Snort function as NIDS, other rule languages such as YARA [44] and Sigma [31] function in different operational contexts and use different syntax. While these differences may limit the transferability of findings, the insights provided in this work retain relevance since the starting point (i.e., benign and malicious tests) and the goal (i.e., detecting malicious behavior) remain similar. We believe that SURICAP can be extended or used as a blueprint to study related different rule languages or different operational contexts, such as those offered by Zeek [25], YARA, or Sigma.

¹⁰Note that addition of a single character (e.g., *!*) may reverse the detection logic.

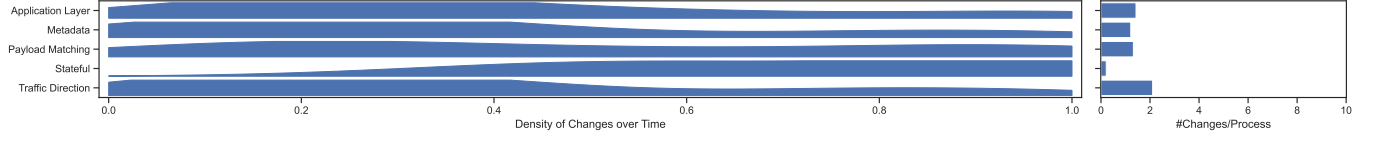


Figure 7: Summary of expert processes showing changes made over time (left), and number of changes per category (right).

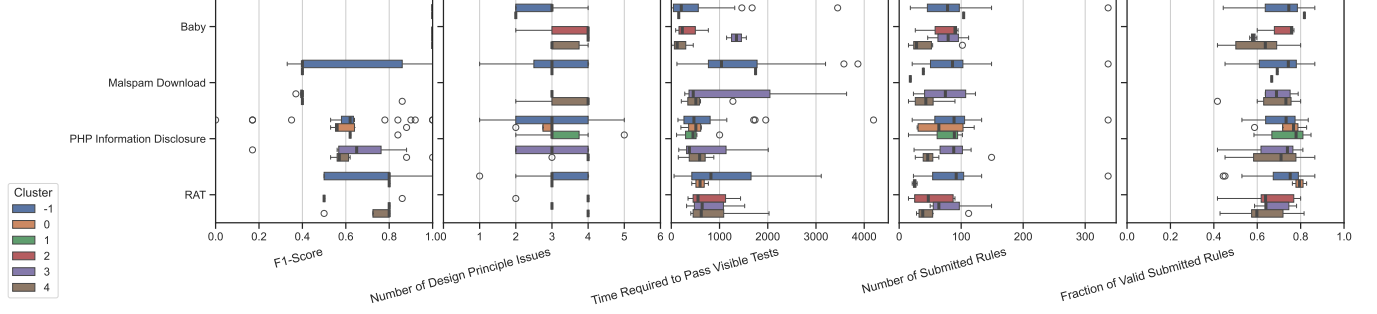


Figure 8: Comparison of the five clusters and the rejection cluster within each scenario.

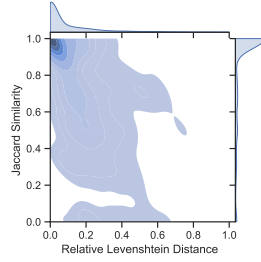


Figure 9: Edit distances of rule updates.

Although the subjects of our study are primarily novices in detection engineering, which are not most representative of typical SOC rule engineers, we believe that our findings present meaningful insights that are applicable to SOC operations since experience does not significantly affect rule engineering processes according to our findings in Section 4. Moreover, we compared the rule engineering processes of several SOC experts in Section 4 and found them to be similar to those of most processes in our study. Previous research also studied novices to better understand security-related tasks [6, 15]. Furthermore, we remark that many rules (specifically community-sourced) are not necessarily written by experienced rule engineers as discussed in Section 2 and that typical SOC hires have experience-levels similar to our participants [16].

While our work addresses the rule engineering aspect of detection engineering, the processes detection engineers follow to select detection characteristics (i.e., Phase 1) is not captured by SURICAP. Although we acknowledge that characteristics chosen significantly impact coverage and specificity [41], studying how detection engineers select characteristics would be intrusive (i.e., employing screen captures) or require significant efforts in developing data collection methods for interactions with applications like Wireshark, TCPDUMP, or web browsers. Although other research methods could be applied, they complicate data collection at-scale,

which is preferred due to the large variety in rule engineering strategies as discussed in Section 4. Nevertheless, Phase 1 and may be an interesting topic for future research given that rule engineers rarely pivot to alternative rule designs as discussed in Section 4.

While the presented data analysis is limited, we shed light on the effects on rule engineering processes of prior experience, training, and test data availability. Enabling future research, we release the PCAPs used to evaluate rules, the platform used to organize CTF-style workshops, the collected datasets, and the code used to conduct the presented analysis. We believe the integrated development environment and continuous feedback provided by SURICAP may be beneficial for training or otherwise improve rule engineering processes similar to what is typically offered by cyber ranges [13, 43]. To support such endeavors, we release SURICAP¹¹ to the public.

6 Conclusion

This work presents SURICAP as a novel platform to study NIDS rule engineering and used it to collect data, from which we derive the first academic characterization of detection engineering processes. We identified rule engineering phases, a common implementation strategy, and shed light on the importance of available test data and fine-tuning behavior, to benefit researchers and SOCs.

Acknowledgments

The authors thank colleagues at other institutions who helped organize the workshops, and everyone who participated in them. This paper was edited for grammar using LanguageTool and Writefull. This publication is part of the CATRIN and INTERSECT projects (with numbers NWA.1215.18.003 and NWA.1160.18.301), which is (partly) financed by the Dutch Research Council (NWO).

¹¹ <https://github.com/Koen1999/suritrain>

References

- [1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. 2001. On the surprising behavior of distance metrics in high dimensional space. In *Database Theory — ICDT 2001*. Springer, Berlin, Heidelberg, 420–434.
- [2] Bushra A. Alahmadi, Louise Axon, and Ivan Martinovic. 2022. 99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms. In *USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 2783–2800.
- [3] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. 2023. Humans vs. Machines in Malware Classification. In *32nd USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 1145–1162.
- [4] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *31st USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 3971–3988.
- [5] Broadcom. 2025. Many branches in the AsyncRAT tree. <https://www.broadcom.com/support/security-center/protection-bulletin/many-branches-in-the-asyncrat-tree>
- [6] Kevin Burk, Fabio Pagani, Christopher Kruegel, and Giovanni Vigna. 2022. Decomposition: How Humans Decompile and What We Can Learn From It. In *31st USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 2765–2782.
- [7] Cisco. 2025. Snort. <https://www.snort.org/>
- [8] Cisco. 2025. Snort Rule Subscriptions. <https://www.snort.org/products>
- [9] MITRE Corporation. 2024. MITRE ATT&CK®. <https://attack.mitre.org/>
- [10] Luciano da F. Costa. 2021. Further Generalizations of the Jaccard Index. <https://arxiv.org/abs/2110.09619>
- [11] Wireshark Foundation. 2025. Wireshark. <https://www.wireshark.org/>
- [12] Sebastian Garcia. 2025. Malware Capture Facility Project. <https://stratosphereips.org>
- [13] Magdalena Glas, Manfred Vielberth, Tobias Reittinger, Fabian Böhm, and Günther Pernul. 2022. Visual programming in cyber range training to improve skill development. In *International Symposium on Human Aspects of Information Security and Assurance*. Springer International Publishing, Cham, Switzerland, 3–13.
- [14] Travis Green. 2024. Suricata Style Guide. <https://github.com/sidallocation/suricata-style-guide>
- [15] Leon Kersten, Santiago Darré, Tom Mulders, Emmanuele Zambon, Marco Caselli, Chris Snijders, and Luca Allodi. 2024. A Security Alert Investigation Tool Supporting Tier 1 Analysts in Contextualizing and Understanding Network Security Events. In *2024 Annual Computer Security Applications Conference*. IEEE, Piscataway, NJ, USA, 890–905.
- [16] Kathryn Knerler, Ingrid Parker, and Carson Zimmerman. 2022. 11 Strategies of a World-Class Cybersecurity Operations Center. MITRE Corporation, McLean, VA. <https://www.mitre.org/news-insights/publication/11-strategies-world-class-cybersecurity-operations-center>
- [17] Faris Bugra Kokulu, Ananta Soneji, Tiffany Bao, Yan Shoshitaishvili, Ziming Zhao, Adam Doupe, and Gail-Joon Ahn. 2019. Matched and Mismatched SOC's: A Qualitative Study on Security Operations Center Issues. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, New York, NY, USA, 1955–1970.
- [18] Alessandro Mantovani, Simone Aonzo, Yanick Fratantonio, and Davide Balzarotti. 2022. RE-Mind: a First Look Inside the Mind of a Reverse Engineer. In *31st USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 2727–2745.
- [19] Leland McInnes, John Healy, Steve Astels, et al. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* 2, 11, Article 205 (2017), 2 pages.
- [20] Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. <https://arxiv.org/abs/1802.03426>
- [21] Jack Naglieri. 2022. Think Like a Detection Engineer, Pt. 2: Rule Writing. <https://panther.com/blog/think-like-a-detection-engineer-pt-2-rule-writing>
- [22] OISF. 2023. Suricata. <https://suricata.io/>
- [23] OISF. 2023. Suricata User Guide. <https://docs.suricata.io/en/latest/>
- [24] Blessing Onyegbula. 2023. Detection Engineering Explained. https://www.splunk.com/en_us/blog/learn/detection-engineering.html
- [25] The Zeek Project. 2023. Zeek. <https://zeek.org/>
- [26] Proofpoint. 2024. ET Category Descriptions. <https://tools.emergingthreats.net/docs/ETProRuleCategories.pdf>
- [27] Proofpoint. 2025. Emerging Threats Forum Rule Signatures. <https://community.emergingthreats.net/c/rule-signs/11>
- [28] Proofpoint. 2025. ET Pro® Ruleset. <https://www.proofpoint.com/us/threat-insight/et-pro-ruleset>
- [29] Adriana Radu, Leon Kersten, Rik Wosyka, Tom Mulders, Emmanuele Zambon, and Luca Allodi. 2025. A Test Tool to Evaluate the Skill Sets of Tier-1 Security Analysts in a SOC Environment: A Case Study from Recruitment to Operations. In *Proceedings 2025 Workshop on Security Operation Center Operations and Construction*. Internet Society, Reston, VA, USA, 8 pages.
- [30] Chitta Ranjan, Samaneh Ebrahimi, and Kamran Paynabar. 2022. Sequence graph transform (SGT): a feature embedding function for sequence data mining. *Data Mining and Knowledge Discovery* 36, 2 (2022), 668–708.
- [31] SigmaHQ. 2025. Sigma Rules. <https://sigmahq.io/docs/basics/rules.html>
- [32] StatCounter. 2025. Desktop Operating System Market Share Worldwide. <https://gs.statcounter.com/os-market-share/desktop/worldwide>
- [33] Sathya Chandran Sundaramurthy, John McHugh, Xinming Ou, Michael Wesch, Alexandru G. Bardas, and Siva Raj Rajagopalan. 2016. Turning Contradictions into Innovations or: How We Learned to Stop Whining and Improve Security Operations. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Berkeley, CA, USA, 237–251.
- [34] Christian Taillon. 2024. Snort and Suricata Rule Guide. <https://christian.io/snort-suricata-guide>
- [35] André Tavares. 2024. Hunting PrivateLoader: The malware behind InstallsKey PPI service. <https://www.bitsight.com/blog/hunting-privateloader-malware-behind-installskey-ppi-service>
- [36] OWASP Cheat Sheets Series Team. 2025. XSS Filter Evasion. https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html#url-encoding
- [37] Koen Teuwen. 2025. suricata-check. <https://suricata-check.teuwen.net/>
- [38] Koen T.W. Teuwen, Tom Mulders, Emmanuele Zambon, and Luca Allodi. 2025. Ruling the Unruly: Designing Effective, Low-Noise Network Intrusion Detection Rules for Security Operations Centers. In *Proceedings of the 2025 ACM on Asia Conference on Computer and Communications Security*. ACM, New York, NY, USA, 1428–1441.
- [39] Truesec. 2025. Detection Engineering – A Strategic Imperative. <https://www.truesec.com/security/detection-engineering-a-strategic-imperative>
- [40] Mathew Vermeer, Natalia Kadenko, Michel van Eeten, Carlos Ga nán, and Simon Parkin. 2023. Alert Alchemy: SOC Workflows and Decisions in the Management of NIDS Rules. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, New York, NY, USA, 2770–2784.
- [41] Mathew Vermeer, Michel van Eeten, and Carlos Ga nán. 2022. Ruling the Rules: Quantifying the Evolution of Rulesets, Alerts and Incidents in Network Intrusion Detection. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ACM, New York, NY, USA, 799–814.
- [42] Manfred Vielberth, Fabian Böhm, Ines Fichtinger, and Günther Pernul. 2020. Security Operations Center: A Systematic Study and Open Challenges. *IEEE Access* 8 (2020), 227756–227779.
- [43] Manfred Vielberth, Magdalena Glas, Marietheres Dietz, Stylianos Karagiannis, Emmanouil Magkos, and Günther Pernul. 2021. A digital twin-based cyber range for SOC analysts. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer International Publishing, Cham, Switzerland, 293–311.
- [44] VirusTotal. 2025. YARA Documentation. <https://yara.readthedocs.io/en/latest/>

A Instruction contents

The preparatory instruction first covers the experiment design (including the existence of hidden tests), and the general experience the participants will undergo. Practical aspects of Wireshark [11], a tool to inspect network traffic captures, are covered by discussing common features such as protocol recognition, display filters, and the following of streams. Subsequently, the concept of intrusion detection is generally introduced and Suricata is covered in-depth by explaining the rule syntax and highlighting common functional keywords including content, file.data, flow, flowbits, and threshold as well as transformations, modifiers, and protocol-specific keywords for DNS, HTTP, and TLS. The existence of other options is emphasized with reference to the official Suricata documentation [23], which is also accessible through the platform UI. We exemplify how a simple rule containing all keywords can be written. The instruction also explains the various scenarios that users tackle during the activity, highlighting why one would want to detect such scenarios and possibly relevant information without prescribing detection characteristics. The general instruction ends with practical details on the employed platform and the leaderboard scoring method, as well as a live demonstration of the platform. The general instruction takes approximately one hour.

B Intake Questionnaire

To grade the intake questionnaire, we group questions of the same topic. Furthermore, we distinguish two types of multiple choice questions: (1) *single-answer questions* to which only a single answer can be selected, and (1) *multiple-answer questions* to which multiple answers may be selected and that have zero or more valid answers. For single-answer questions, participants receive a score of one if the correct answer is selected and zero otherwise. For multiple-answer questions, the score equals the number of selected correct answers minus the number of incorrect answers, with a minimum of zero. The contents of the intake questionnaire are found hereafter.

Questions 1–4 collected the participant's contact address, leaderboard username, confirmation of having the legal age to consent, and the informed consent.

Question 5: How did you first learn about the CTF activity?

- (a) PROMOTION CHANNEL 1 (E.G., COURSE)
- (b) PROMOTION CHANNEL 2 (E.G., CTF ASSOCIATION)
- (c) ...

Question 6: What is your gender?

- (a) Male (b) Female (c) Other (d) I prefer not to disclose

Question 7: What is the highest level of education that you have completed?

- (a) Primary education (ISCED 1)
- (b) Lower secondary education (ISCED 2)
- (c) Upper secondary education (ISCED 3)
- (d) Post-secondary non-tertiary education or Short-cycle tertiary education (ISCED 4 or 5)
- (e) Bachelor's or equivalent (ISCED 6)
- (f) Master's or equivalent (ISCED 7)
- (g) Doctorate or equivalent (ISCED 8)

Question 8: What is your level of expertise regarding Capture the Flags (or similar activities)?

- (a) I have never heard of a Capture the Flag.
- (b) I know what a Capture the Flag is but have never participated in one.
- (c) I have previously participated in a Capture the Flag.
- (d) I have participated in multiple Capture the Flags.
- (e) I have previously organized a Capture the Flag.

Question 9: Do you have any other relevant experiences (such as a job related to Security Operations or high expertise in another area you deem relevant) that may be relevant for the research? If so, describe the topic and the relevant experience below.

Question 10: How does the TCP protocol ensure that the packets are delivered in the correct order?

- (a) Through the use of SYN and ACK flags.
- (b) Through the encryption and decryption of data packets.
- (c) Through the use of sequence and acknowledgment numbers.
- (d) Through the use of checksums.
- (e) I do not know.

Question 11: What types of information can be exchanged during the initiation of a TLS session?

- ☐ Plain-text HTTP requests
- ☐ Server and client Hello messages
- ☐ Encrypted handshake messages
- ☐ Certificate
- ☐ I do not know.

Question 12: Which of the following statements regarding network scanning are true?

- ☐ Lack of response (SYN,ACK) to a TCP SYN implies a port is closed.

- ☐ If a vulnerability scanner detects a vulnerability, the scanned device has an exploitable vulnerability.
- ☐ Nmap can only perform UDP/TCP scans.
- ☐ If you want to know which ports may be open on a public-facing IP address, you must scan it.
- ☐ Network scanning may be used to inspect application layer features such as used software and versions.
- ☐ I do not know.

Question 13: Which of the following statements regarding Cross-Site Scripting (XSS) are true?

- ☐ During a successful XSS attack malicious code is executed on a server distributing content.
- ☐ XSS attacks may be prevented through input sanitization.
- ☐ XSS attacks can be used to steal credentials.
- ☐ During a successful XSS attack the attacker may execute malicious code within the web browsers of other users visiting a website.
- ☐ XSS attacks are considered to be a subset of SQL injection attacks.
- ☐ I do not know.

Question 14: Which of the following statements regarding Persistence are true?

- ☐ Persistence always implies that following a malware infection, malware will remain present after reinstallation of the operating system.
- ☐ Persistence must be obtained in order for malware to accomplish its goal.
- ☐ In order to obtain persistence, malware must add an executable file to the startup folder on Windows.
- ☐ Metasploit is a tool that may be used to obtain persistence depending on the targeted system.
- ☐ Persistence always has associated network traffic.
- ☐ I do not know.

Question 15: What is/are (a) feature(s) offered by Wireshark?

- ☐ Filtering of packets
- ☐ Decoding of transmitted data
- ☐ Blocking network traffic
- ☐ Decrypting all encrypted network traffic
- ☐ Producing network I/O statistics
- ☐ Inspecting raw bytes transmitted in streams
- ☐ I do not know.

Question 16: What is the purpose of the `ip.addr==192.168.178.1/24` filter in Wireshark?

- (a) To only show network packets originating from the subnet 192.168.178.1/24
- (b) To only show network packets sent to the subnet 192.168.178.1/24
- (c) To only show network packets originating from or sent to the subnet 192.168.178.1/24
- (d) To only show network packets originating from and sent to the subnet 192.168.178.1/24
- (e) It is an invalid filter.
- (f) I do not know.

Question 17: This question is just to check if you are actually reading questions carefully. It is an attention check. If you read this, please click the third option.

(a) HTTP (b) DNS (c) SSH (d) TLS (e) UDP (f) TCP (g) I do not know.

Question 18: Which of the following statements regarding different intrusion detection paradigms are true?

- ☐ Signature-based intrusion detection methods always rely on Atomic Indicators of Compromise (IOCs).
- ☐ Anomaly-based intrusion detection methods always rely on machine learning.
- ☐ Signature-based intrusion detection methods primarily rely on knowledge of malicious behaviors.

- ☐ Anomaly-based intrusion detection methods primarily rely on knowledge of benign behaviors.
- ☐ Anomaly-based intrusion detection methods are a subset of signature-based intrusion detection methods.
- ☐ I do not know.

Question 19: What are built-in functionalities offered by Suricata?

- ☐ Matching bytes at specific locations
- ☐ Stateful detection across different flows in which the same IP address is involved
- ☐ Matching specific traffic directions
- ☐ Decoding of certain HTTP buffers
- ☐ Stateful detection within the same flow
- ☐ Matching using a remote API
- ☐ Matching using regular expressions
- ☐ I do not know.

Question 20: Which of the following buffers would be matched by the following sequence of Suricata options: CONTENT:"FOO"; CONTENT:"BAR"; DEPTH:5;

☐ FOObar ☐ barFOO ☐ foobar ☐ barfoo ☐ I do not know.

Question 21: Read the following Suricata documentation and select the factually correct statements one can derive from the given sentence.

[SURICATA DOCUMENTATION ON *STARTSWITH* AND *DOTPREFIX*]

Now consider a typical DNS request to *google.com*. Which of the following rules are valid rules and will match this request?

- ☐ alert dns any any → any 53 (msg:"DNS Request to google.com"; dns.query; content:"google.com"; startswith; sid:1;)
- ☐ alert dns any any → any 53 (msg:"DNS Request to google.com"; dns.query; content:".google.com"; startswith; sid:1;)
- ☐ alert dns any any → any 53 (msg:"DNS Request to google.com"; dns.query; dotprefix; content:"google.com"; startswith; sid:1;)
- ☐ alert dns any any → any 53 (msg:"DNS Request to google.com"; dns.query; dotprefix; content:".google.com"; startswith; sid:1;)
- ☐ alert dns any any → any 53 (msg:"DNS Request to google.com"; dns.query; content:"google.com"; startswith; dotprefix; sid:1;)
- ☐ alert dns any any → any 53 (msg:"DNS Request to google.com"; dns.query; content:".google.com"; startswith; dotprefix; sid:1;)
- ☐ I do not know.

C Additional scenarios

Baby. To familiarize participants with Suricata and the CTF platform, and similarly to [6], participants first encounter a *Baby* ‘warm-up’ scenario. This scenario requires participants to detect a DNS request to *DETECTME.FOOBAR*. We manually collected a PCAP for this scenario using *NSLOOKUP*, which resulted in one A and one AAAA query. We instruct the participants to engineer a rule that detects DNS requests to this domain. Whereas simple, it allows participants to familiarize with the interface and provides the opportunity for a quick practical refresher on Suricata syntax.

Malspam Download. This scenario covers the retrieval of a webpage that initiates a download for a malicious binary from the *Guildma/Astaroth* campaign. We leverage four PCAPs containing HTTP traffic following the clicking through on malspam emails from this campaign, collected over a time span of over two years. We present participants with the oldest capture¹² for the visible tests. We used the three most recent captures^{13,14,15} for invisible

tests to assess the generalizability of rules to more recent incidents of the same campaign. Although all four captures include similar mechanisms to initiate a download, the most recent PCAP contains double base64-encoded HTML to initiate a download, indicating adversaries have become increasingly evasive over time. The first three PCAPs exhibit very similar behavior although specifics such as domain names, payloads, and some irrelevant data vary.

RAT. The *RAT* scenario covers the outbound C2 communications of a Remote Access Toolkit (RAT) of the AsyncRAT family. These communications occur over TLS, making it more challenging than the other scenarios. Nevertheless, the communications preceding the encrypted TLS traffic offer various detection opportunities, amongst others due to the use of unusual certificates. We leverage two network traffic captures from 2024^{16,17} and an older capture of the AsyncRAT variant that we had previously collected in 2022 for educational purposes. The most recent capture contains traffic from *DcRat*, which is a variant of AsyncRAT [5], amongst others varying certificate details. Similarly to the other scenarios, we use the oldest available PCAP as visible test for the participant, and the more recent PCAPs as invisible tests to assess the generalizability of rules to more recent incidents corresponding to the same malware family.

D Experience effects regression

Figure 10 shows the regression¹⁸ using a Robust Linear Model ($p = 0.05$) over the participants’ experience to predict their performance as described and discussed in Section 4.

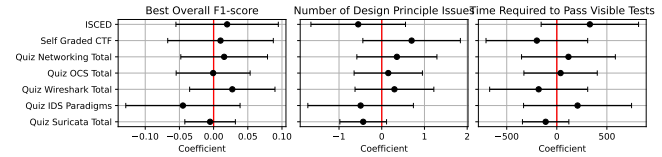


Figure 10: Three plots depicting the relation between the experience and their performance across all scenarios, each showing the coefficient together with confidence intervals.

E Progression through rule engineering phases

Figure 11 shows the number of participants in a certain phase.

F Detailed cluster overviews

The five clusters/strategies of rule engineering processes resulting from and described in Section 4 are shown in Figure 12¹⁹. Figure 13 contains the detailed overview of the clusters described in Section 4 similar to how the expert cluster is depicted in Figure 7

¹²<https://www.malware-traffic-analysis.net/2021/04/12/index.html>

¹³<https://www.malware-traffic-analysis.net/2021/07/02/index.html>

¹⁴<https://www.malware-traffic-analysis.net/2022/09/21/index.html>

¹⁵<https://www.malware-traffic-analysis.net/2023/12/11/index.html>

¹⁶<https://www.malware-traffic-analysis.net/2024/01/09/index.html>

¹⁷<https://www.malware-traffic-analysis.net/2024/03/14/index.html>

¹⁸All VIF scores are below 2.

¹⁹UMAP [20] is used with the Manhattan distance to reduce to 2-dimensions.

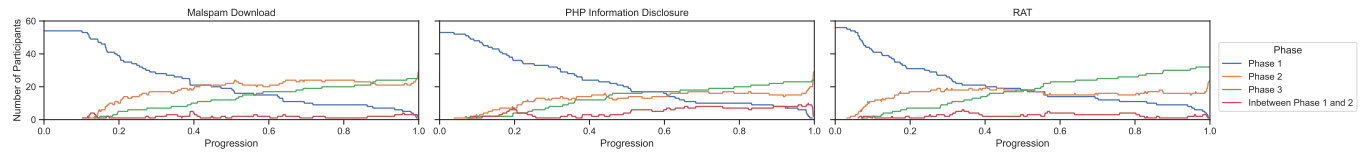


Figure 11: Participants per rule engineering phase over time for each main scenario.

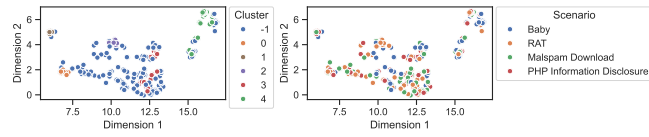


Figure 12: Clusters of rule engineering processes where, the left highlights the cluster a process belongs to, and the right highlights the scenario a process was for.

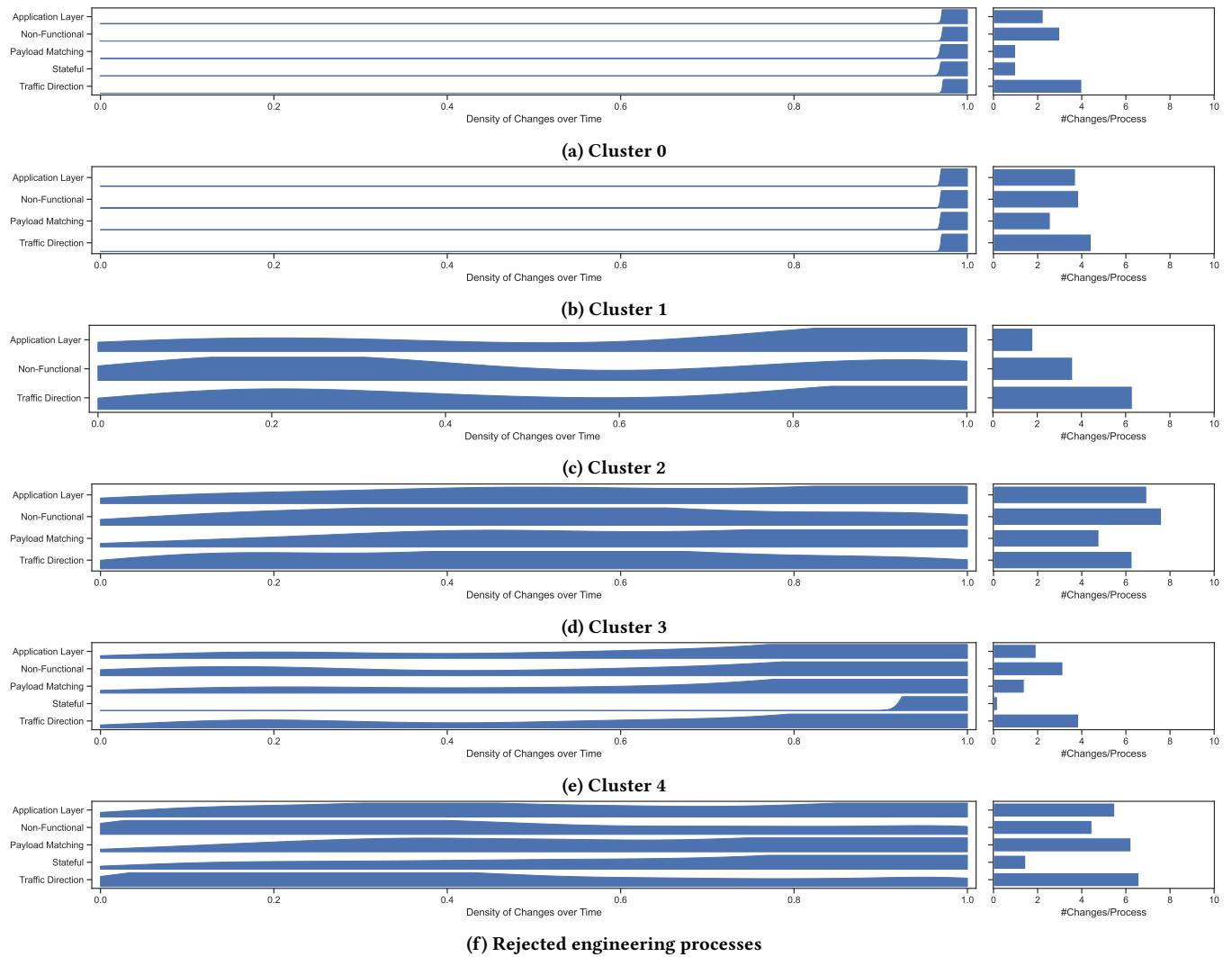


Figure 13: Detailed overview of the five clusters of engineering processes and the group of rejected engineering processes, depicting within each subfigure, the changes made over time (left), and the number of changes per category (right).