

Characterizing the Modification Space of Signature IDS Rules

Ryan Guide*, Eric Pauley†, Yohan Beugin†, Ryan Sheatsley†, Patrick McDaniel†

*The Pennsylvania State University, †University of Wisconsin-Madison

*rvg106@psu.edu, †{epauley, ybeugin, sheatsley, mcdaniel}@cs.wisc.edu

Abstract—Signature-based Intrusion Detection Systems (SIDSs) are traditionally used to detect malicious activity in networks. A notable example of such a system is Snort, which compares network traffic against a series of rules that match known exploits. Current SIDS rules are designed to minimize the amount of legitimate traffic flagged incorrectly, reducing the burden on network administrators. However, different use cases than the traditional one—such as researchers studying trends or analyzing modified versions of known exploits—may require SIDSs to be less constrained in their operation. In this paper, we demonstrate that applying modifications to real-world SIDS rules allow for relaxing some constraints and characterizing the performance space of modified rules. We develop an iterative approach for exploring the space of modifications to SIDS rules. By taking the modifications that expand the ROC curve of performance and altering them further, we show how to modify rules in a directed manner. Using traffic collected and identified as benign or malicious from a cloud telescope, we find that the removal of a single component from SIDS rules has the largest impact on the performance space. Effectively modifying SIDS rules to reduce constraints can enable a broader range of detection for various objectives, from increased security to research purposes.

I. INTRODUCTION

Network intrusion detection systems (NIDS) are a component of security that exist to protect networks against adversaries. Signature-based intrusion detection systems (SIDS) are a distinct category of NIDS that utilize known signatures to defend against exploits. While other NIDSs react to anomalous activity, SIDSs have precisely defined rules to identify specific attacks [1]. An example of one such system is Snort: a free, open source, and lightweight SIDS introduced by Roesch [2]. Snort rules can be written by anyone and the official Snort website [3] publishes the latest detection rules on a regular basis. As the space of threats continues to evolve, Snort rules are written in response to protect against adversaries.

Current systems are fine-tuned to minimize false positives, which occur when benign traffic is flagged as malicious. False flagging benign traffic is harmful to critical networks, as it disrupts availability and can potentially cause customers to lose trust in the service [4], [5]. As a result, rules have been tailored to fit the semantics of specific attacks, constraining at the same time SIDSs to their traditional security use case. However, if one wanted to achieve different detection objectives, they would not readily be able to do so. The applications of expanding the performance space of SIDSs extend to researchers who study specific regions of traffic: whether that be through eliminating all false positives to look at only malicious traffic, or by disregarding false positives and seeking to capture all malicious traffic.

In this paper, we aim to loosen the constraints on SIDS rules through evaluating modifications to the rules. SIDS rules are composed of conditions that must be all met in order for traffic to be matched. Since additions of conditions only serve to further constrain rules, we focus our attention solely on removals.

Since the full space of removals is intractable, we develop an iterative technique to maneuver within the space for the corresponding desired outcome. We gradually remove parts of the initial rules and compare the receiver operator characteristic (ROC) curves of the modified rules to a heuristic ground truth. We then identify the Pareto frontier [6] of rule performance and repeats our process further until either no further modifications can be made, or the Pareto frontier stops expanding.

We evaluate the modifications made to the rules on the detection performance by using traffic collected and identified as benign or malicious from the DSCOPE cloud telescope [7]. We find the Pareto frontier reachable through removals and identify which and why modifications are the most efficient. We find that single option removal from each rule has the greatest impact, expanding the area under the ROC curve by 11.3%. Of the removals, we identify the cause of change in detection.

Through removals, we demonstrate how to loosen constraints on SIDS rules. This enables a broader range of detection for various objectives, from increased security to research purposes.

II. BACKGROUND

An intrusion detection system (IDS) is a system that operates on a network with the intent of blocking malicious traffic while allowing benign traffic to pass through unhindered. IDSs can be divided into three groups: signature based intrusion detection systems, anomaly based systems, and stateful protocol analysis [1]. We focus on signature based intrusion detection systems (SIDS).

a) Snort: Snort is an open source SIDS that consists of a collection of rules that are checked against network traffic for matching packets. When a match is found, an alert is triggered [2]. The discovery of new exploits require new detection rules to be written. Our study focuses on Snort 3 [3]. Snort efficiently performs signature and packet comparison in four steps: (1) packet decoding and pre-processing, (2) fast pattern matching, (3) finer rule matching and packet inspection for traffic flagged by previous step, and (4) final logging and triggering of actions (blocking, alerts, etc.). In our study, none of the core functionalities of Snort is modified, only the rules themselves.

b) Snort Rules: A Snort rule consists of a header and a body. The header indicates the protocol, source, direction,

and destination of the packets to compare against the rule as well as the action to take in case of a match. The body is composed of the different rule options for matching. As soon as a packet fails to meet an option, Snort stops reading the rule [8]. The `content` option is one of the primary tools used to match packets; the packet is scanned for the pattern specified in the `content` option. There is no limit as to how many `content` options can be included in a rule. Other modifiers can also be applied to specify how sensitive the matching should be, and the target section of the packet like `http_header`. These modifiers exist to speed up matching and reduce false positives.

III. METHODOLOGY

The current rules are tailored to specific network attacks to reduce false positives as they can harm availability of critical networks [4], [5]. We seek to make these rules less rigid for other use cases that for instance extend to researchers who study specific regions of traffic: whether that be through eliminating all false positives to look at only malicious traffic, or by disregarding false positives and seeking to capture all malicious traffic.

We are interested in measuring the impact of modifying the rules on the performance of the detection of malicious traffic. We observe that the addition of options to existing rules only further constraint them and decrease detection, as a result, we propose to evaluate the removal of such options. We expect that this will increase the amount of traffic that is detected. Among the different rule options, the *general* options—as defined by the Snort documentation [8]—have no impact on the performance other than semantics, making removal pointless. In our study, we use a finite set of rules containing 60 distinct options that can be removed without error. We also assume we have access to a data set, discussed in section IV of benign and malicious traffic to assess the performance of rules.

a) The Removal Function: We formally define the body of a rule R as a conjunction of n removable options a_i .

$$R = a_1 \wedge a_2 \wedge \dots \wedge a_n = \bigcap_{i=1}^n a_i$$

To gain an understanding of how we are removing options from rules, we define a function f to gather the single removals for each rule: $f(R) = \{R_i \mid R_i = R \setminus \{a_i\}, 0 < i < n\}$ The set of removals can be defined as all R_i , where i is index of the option being removed. Since removing all options renders the rule trivial, we omit this possibility. As such, if $R = a_1$, $f(R) = \emptyset$. Generalizing this for more than one removal, we get the function $f(R) = \{R_{i_k} \mid R_{i_k} = R - \{a_{i_1}, \dots, a_{i_k}\}, 0 < i_1 < \dots < i_k < n\}$ With n is the number of total removable options, i is the option being removed, and $0 < k < n$ is the number of options being removed.

b) Calculating the Space of Modifications: The total number of possible removals can be calculated by counting combinations. Let i be the number of options removed, and let n be the total number of options in a rule. Let k represent all rules, and let k_i be the total number of rules with i removable options, for $0 < i < n$. We then have k represented as $k = k_1 \cup k_2 \cup \dots \cup k_n$. We calculate all possible combinations of removals from each subset. For each subset k_i , we want to calculate removing

Option	Instances	Option	Instances
content	101785	pcre	10823
flow	42205	flowbits	10330
service	37984	http_uri	10248
file_data	20833	byte_test	4368

TABLE I
NUMBER OF INSTANCES FOR THE MOST COMMON OPTIONS.

$0 < j < i$ options from the rules. Thus, we get the following: $k_i = \sum_{j=1}^{i-1} \binom{i}{j}$ We omit $j = 0$ (the original rules) and $j = i$ (removing all parts). We calculate each k_i with as $2^i - 2$. For all $0 < i < n$, we get the following summation $\sum_{i=1}^{n-1} 2^i - 2 * k_i$

We find that we had over 137 billion possible combinations of removals, which is intractable. To handle this, we applied some limits to our study, as addressed in the next paragraph. This also enabled us to study the results of removals more closely. We can observe monotonic changes between additional removals. Removing n options at once without knowing the behavior of removing each option individually leads to wild speculation instead of concrete analysis.

c) Iterative Exploration: We develop a heuristic to explore iteratively the space of possible removals; to start, we run the original rules on our traffic data and plot them on a ROC curve. We then consider each individual option removal on all the rules, run the traffic detection again on these, and add the corresponding performance points to the ROC curve obtained so far. We then compute the convex hull of all points using the Graham Scan algorithm [9] to obtain the Pareto frontier reachable at that stage. If the Pareto frontier does not expand past the performance of the previous stage, we stop modifications. Otherwise, for each point on the Pareto frontier, we create additionally perform removals on the corresponding rules, and repeat the process described previously. Adopting this heuristic effectively reduces the exploration of modifications to a tractable space, indeed after analyzing the set of i removals from the rules, we no longer have to consider another $\binom{n}{i+1}$ removal combinations. Instead, by focusing on the k points along the frontier, we perform only $= k \binom{n-i}{1}$ additional removals.

IV. EVALUATION

The Snort rules we use in our experimentation are taken from March 4, 2022 from the Snort website [3]. To evaluate the performance of rules, we need two key components: traffic to run the rules on and a means to determine which traffic is malicious.

a) The Rule Set: Overall we have 42893 rules in our study. There are 60 distinct removable options present in our rules. We show the most commonly occurring options found in our rules in Table I. The maximum number of options present in any rule is 36, and the minimum is 1. The average number of removable options per rule is 5.93, and the median number of removable options per rule is 6. In over half the rules, the `content` option is used more than once. We refer to this subset of rules as the *multi-content rules*.

b) Defining a Ground Truth: We need a set of traffic to establish a ground truth, which was done with two data-sets: a

collection of traffic from the cloud, and a blocklist of IPs. Our traffic data comes from the DSCOPE network telescope, which uses public cloud IP addresses to deploy an interactive Internet telescope [7]. Network telescopes are ranges of IP addresses used to collect traffic [10], [11]. In our case, the telescope was set on servers in the AWS US-east-1 region. The second data-set determined which traffic was malicious. This set was the fireHOL level 4 blocklist of IPs [12]. These IPs were flagged due to scanning activity. Our data from the cloud gives us every single IP that communicated with the network over a twenty-four hour period. Using the IPs from the cloud, we determine which IPs appeared in the blocklist. Since IPs on the blocklist are considered malicious due to scanning activity rather than using an actual exploit, we consider these IPs benign. This may seem counter-intuitive, but scanning traffic is not inherently malicious. While scans can look for devices to infect, if there is no exploit, then it is not malicious. Hence Snort would not flag it as malicious. The IPs monitored were not allocated, so it follows that the traffic was not for legitimate services and can be classified as malicious [7]. We gather the source IP addresses from collected traffic. This yields 43.3 k IP addresses communicating with our telescope. Using the blocklist, we find 37.6 k malicious IPs and 5.6 k benign ones.

c) Creating a Pipeline: In order to efficiently analyze large sets of data, we create a pipeline to manage the flow of data and speed up the entire process. The pipeline consists of four phases: parsing, pre-processing, rule selection, and output. The pipeline runs in the following phases:

- **Parsing:** Using original rules, our parser generates every possible removal of n options, then filters down to the desired subsets for examination. The rules are then run on the cloud data.
- **Pre-processing:** The output from Snort is parsed to create an efficient mapping of the traffic matched by the new rules.
- **Rule Selection:** Given an option, the selector uses mappings to identify rules that have that option removed along with the original rules. This enables the analysis of specific option removals. Not giving an option results in all rules being used.
- **Output:** The pipe identifies which source IPs were found by the rules selected in the previous phase alongside the original rules. The blocklist is used to determine the rates of true and false positives.

d) The Original Rules: Using the ground truth from the FireHOL blocklist, we identify where the original set of rules are on the ROC curve. For the original set of rules, we find that the true positive rate is 44.54%, and the false positive rate is 7.79%.

e) The Pareto Frontier: We run four iterations of removals. For each option that is removed, we run the modified rules with that option removed alongside the rules unaffected by that removal. For example, to test the removal of `http_header`, we run all the original rules that did not have `http_header` alongside the modified rules that have `http_header` removed. Through that configuration we observe the impact of each removal. We study the overall performance of each modification rather than each individual rule. This enables a better understanding of the rules in general, whereas working with rules at the individual level would only allow us to study specific

Removals	Area of Pareto Frontier(%)
Original (0)	68.37
Single (1)	76.10
Double (2)	76.97
Triple (3)	76.99
Quadruple (4)	76.99

TABLE II
THE AREA COVERED BY EACH ROC CURVE AS A PERCENTAGE

cases. Specific cases are uninteresting in this study as they are too narrow in scope. Running the entire subset together achieves the same result as summing the results of each individual rule.

After plotting each removal, we calculate the Pareto frontier for all the results together to determine if the frontier expanded. At each iteration we calculate the area under the Pareto frontier, as shown in Table II. We see that the greatest expansion comes from the removal of a single option, and once it reaches the fourth iteration, the expansion stops. The first iteration expands the frontier by 11.3% of the original size. The second iteration increases the Pareto frontier by only 1.14%. Due to the small change in difference, we primarily focus our analysis on the first iteration. The original frontier, single removals, and the final curve are all plotted in Figure 1

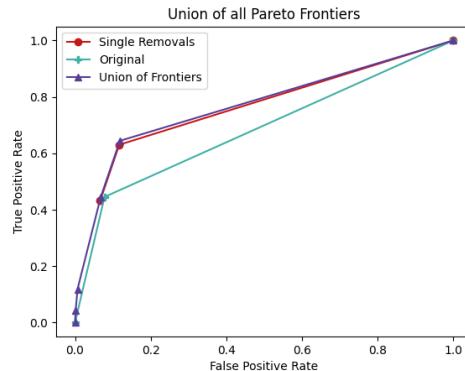


Fig. 1. ROC curves for the original rules, single removals, and the union of all Pareto frontiers.

The points along the Pareto frontier from the fourth iteration were all less than or equal to the frontier points from the third iteration. With the exception of removing `http_header`, `http_uri`, and `flowbits`, the points with lower true and false positives than the original rules were all due to inverting bad classifiers. We discuss the cause for the change in detection in detail later on. **Summary:**

- The greatest change in the frontiers comes from the removal of a single option.
- The Pareto frontier stops expanding after three removals.

f) The Options that Caused Movement: Breaking down our Pareto frontiers further, we look at the specific removals that caused the frontiers to change. To understand the meaning behind the performances, it is necessary to look at the costs of both false positives and false negatives. A false positive disrupts

legitimate traffic, while a false negative allows malicious traffic into the network. To measure the trade off between the two, we define a utility function. Let $\theta \in [0,1]$ be the trade off between false positives (f_p) and false negatives (f_n). We define a cost function, C , as $C = \theta f_p + (1-\theta) f_n$. When $\theta \in [0,0.5]$, reducing false negatives is prioritized. Conversely, when $\theta \in (0.5, 1]$ reducing false positives is prioritized. Naturally $\theta = 0.5$ is equivalent to assigning equal weight to both.

Looking at Figure 2, we see that the minimum cost for balancing false positives and negatives was lower for our set of single removals than the original rules. Up until $\theta = 0.3$, the costs are equivalent. At that point the minimum cost begins to decrease for single removals, but continues to increase for the original rules. At $\theta = 0.87$, the two lines converge again. The area under the curve for the original rules is 0.2064, and the area under the curve for single removals is 0.1792, giving us a total decrease in area by 0.0272. This illustrates how our modifications reduce the overall cost of false positives and negatives.

At both $\theta = 0$ and $\theta = 1$ we see a cost of 0. When $\theta = 0$, the recommended course of action is to block all traffic. Conversely, at $\theta = 1$, the optimal move is to allow all traffic. These are extreme cases and are not likely to be used by enterprises.

For $\theta \in (0, 0.74)$, the modification required to achieve the minimum cost for single removals is to remove a content option from multi-content rules. For $\theta \in [0.74, 0.85]$, the optimal modification is to remove `http_header`. For all other values of θ , the ideal modification lies on the line between the origin and the removal of `http_header`.

For the original rules, we find that the minimum cost is achieved at the original rules exactly for $\theta \in [0.38, 0.84]$. We note that this correlates with the values of θ where we were able to reduce the minimum cost.

For any given detection objective, our rules perform at least as well as the original rules, with the greatest difference occurring from $\theta \in [0.30, 0.70]$. This means that our rules had a lower minimum cost than the original rules when relatively equal weights were applied to the costs of false positives and negatives.

To analyze the impact of each modification, we examine the precision, recall, and the f1-score of single removals [13]. Recall is the percentage of malicious IPs that were detected by a rule set out of the entire malicious IP space. Precision refers to the percentage of malicious IPs found in the set of IPs identified by our rules. The f1-score is a metric that takes the harmonic mean of precision and recall [14]. The f1-score is used as a combination of both precision and recall. Given our class imbalance, we consider the precision, recall, and f1-score for both malicious and benign traffic classification. The top f1-scores are shown in Table III.

To understand why we look at both metrics, we consider the removal of the `flags` option. Upon removal, we find that every single IP address in the space is matched. We call this behavior *universal matching*. Calculating precision and recall for malicious traffic detected gives us a precision of 86.96% and a recall of 100%. However, when we examine benign traffic precision and recall, we get 0% for both. The f1-scores for malicious and benign traffic are 0.9302 and 0, respectively. This demonstrates

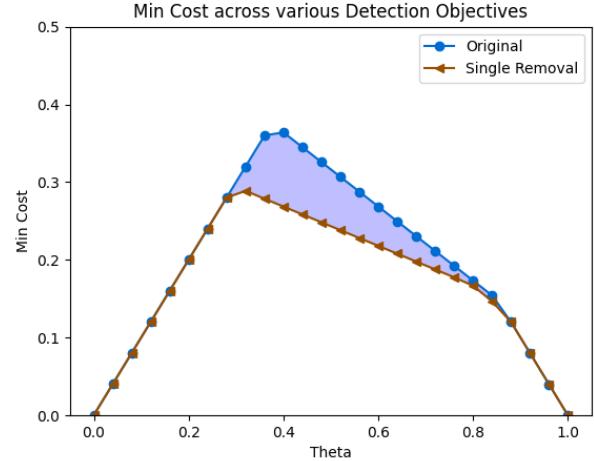


Fig. 2. Minimum cost to achieve ideal detection for various costs on false positives and false negatives for the original rules and single removals. The shaded area represents the area reduced by the modifications.

that while removing `flags` looks like an excellent choice for detecting malicious traffic, we find that it is useless in terms of correctly classifying traffic as benign. By looking at both scores we can determine which removals are actually worthwhile. We also use these scores to find the macro f1-score [15].

First, we look at the original rules. We examine the malicious traffic metric first. The recall is only 44.54%, but the precision is 97.44%, indicating that while it misses roughly half the malicious traffic, it is quite precise. This provides weight to our assertion that the rules are geared towards preventing false positives. We get an f1-score of 0.6113. The cause for this score comes from the lower true positive rate. Looking at the metric for benign traffic, we find a precision of 19.96% and a recall of 92.20% resulting in an f1-score of 0.3282. This is lowered significantly by the low precision. We find a macro f1-score of 0.4697. We find that many of the modifications have very similar scores. We direct our focus in this section to removals from our Pareto frontier and removals with a high macro f1-score.

We now examine notable points from single removals. Here we consider the highest macro f1-scores as well as the points on the frontier for single removals. For each option, we examine the score and factors that influence it. A high f1-score can be misleading, given that most removals saw a high precision for malicious traffic.

Looking at the removal of `http_header`, we see that the recall drops by 1.3%, and the precision increases by .3%. As a result, the f1-score drops to 0.5988. The minor improvement in the ratio of true and false positives comes at a greater cost to recall, and thus worsens the f1-score. We also see a decrease in the f1-score for benign traffic, resulting in an overall drop in the macro f1-score.

Removing a single content option from multi-content rules had 63.03% true positive rate and a false positive rate of 11.67%. The precision of this set decreased by .15% from the original rules. The f1-score was 0.7650 for this set, which reflects the

higher recall than the original rules. The removal of `content` from rules regardless of remaining `content` options did increase the recall by over 50% to 99.78%, but at a cost to precision of over 10%, bringing it down to 86.93%. This resulted in a f1-score of 0.9292. The increase in f1-score comes entirely from the surge in recall. It is important to note that removing `content` resulted in matching 100% of false positives, and since the false positives are greater than the true positives, we would invert this point to achieve a good classifier. This inverted classifier would have a precision of 100%, but a small recall of 0.22%.

The removal of `detection_filter` had the second highest macro f1-score of 0.5567. Removing `detection_filter` gave us a true positive rate of 63.83% and a false positive rate of 26.91%. The increase in true and false positives was 19% for both, indicating a linear increase rather than a bias towards one or the other, hence this point alone did not change the Pareto frontier alone. When combined with the removals of `flow` and `dce_iface`, it created a bad classifier with a higher false positive rate than true positive rate. Upon inverting this classifier, we obtained a new frontier point.

Removing `ip_proto` had the third highest macro f1-score, and was the last single removal to achieve a macro f1-score over 0.5. The score was 0.5169. The true and false positive scores for the removal of `ip_proto` were 52.81% and 11.89% respectively. This was an increase in true positives by 8%, and an increase in false positives by 4%. This improvement is eclipsed by the removal of `content`, and therefore is not included in the frontier for single removals. Similarly to `detection_filter`, it does appear on the union of all frontiers.

The combination of removing both `dce_iface` and `flow` resulted in a bad classifier, but the inverse expanded the Pareto frontier. The inverted removal of both resulted true and false positive rates of 11.85% and 0.66% respectively. The individual removals of `flow` and `dce_iface` were not part of the frontier for single removals. We do note that the removal of `dce_iface` was the cause for the bad classification, as `flow` found more malicious traffic than benign. The f1-scores for both malicious and benign traffic were both quite low, at 0.2117 and 0.2525 respectively. The macro f1-score was 0.2321, which is half the score of the original rules. When `detection_filter` was included the inverted classifier fell to true and false positive rates of 4.19% and 0.16%. The macro f1-score was 0.1593, lowered entirely by the drop in true positives. **Summary:**

- We showed using our utility function that the minimum cost for our modifications is at least as good as the original rules, with the greatest decrease in cost coming from assigning roughly equal costs to false positives and negatives.
- Removing a single `content` from multi-content rules was the most effective modification in terms of macro f1-score.
- While `detection_filter` and `ip_proto` increased detection, we show that there are better alternatives.

g) *The Cause of Movement:* In this section we explain why certain modifications alter detection in the manner that they do and the value in removing each option. The `flags` option checks TCP flags on the packet. Without this filter traffic

Removal	$F1_{malicious}$	$F1_{benign}$	$F1_{macro}$
content (when >1)	0.7650	0.4063	0.5857
<code>detection_filter</code>	0.7605	0.3549	0.5567
<code>ip_proto</code>	0.6832	0.3506	0.5169
<code>flags</code>	0.9302	0.0	0.4651
original rules	0.6113	0.3282	0.4697

TABLE III
F1-SCORES FOR SIGNIFICANT REMOVALS.

more traffic is matched. This increase is arbitrary, and is not particularly insightful. There are only 24 rules in which `flags` was present: less than 1% of all our rules.

The decrease in detection from `http_header` can be explained by the Snort 3 documentation. Since the header is not decoded the detection cursor might not find the matching string. There are 3434 rules with `http_header` present in our rules, meaning the removals affect only 8% of the rules. We see a slight decrease in detection from the removal of `http_header` where the decrease in true positives is slightly larger than the decrease in false positives.

Next we consider the `content` option. As demonstrated, a subset of `content` proved to be one of the optimal removals. The change in `content` can be split into two categories; removing `content` in general and the subset of multi-content rules. Since `content` is the primary tool used for detection, removing it enables more matches. Removing `content` from a multi-content rules greatly increased the range of detection. Since we saw nearly complete matching of all IPs, we classify this increase as random. Even though the inverse of this results in obtaining a small percentage of true positives with no false positives, it is likely this is random noise, and therefore is not taken into consideration. Examining the subset of `content` removals from multi-content rules yields a significantly lower rate of false positives. We instead see that the false positive rate is 11%, while the true positives increased to 63%. As there are still `content` options left in the rule, traffic that matches less of the characteristics is collected. However, this enables capturing of similar malicious traffic. A slight adjustment to the malicious traffic, either repositioning where the exploit is in the payload of the packet or an alteration to the exploit itself can be missed by the original rule. When the `content` option to match that part of the exploit is no longer present, the rule becomes more flexible in detecting variations on the attack. The other key benefit to having more than one `content` option is the repetition it enables. Consider a rule with three `content` options: *A*, *B*, and *C*. Removing a single `content` option gives us three new rules: one with *A* and *B*, one with *B* and *C*, and one with *A* and *C*. Compared to the original rule, we now have three rules which are able to sense variants on the original attack. Similar behavior was observed in another study [16], where they replaced characters in the `content` option string with general characters. From this they found variations on attacks that were missed prior. Out of the 42893 rules present in our experiment, 24769 of them are multi-content rules. This means about 57% of the rules are modifiable in this manner. There are only 344 rules without any `content` options,

meaning 99% of all our rules has at least one content option.

The `detection_filter` option adds a counter to the rule, indicating that it must receive a certain number of communications from an IP before it will allow the rule to fire. Thus removing this logically eliminates the filter and allows the rules to fire more often. As a result, we found that half of the additional traffic flagged was malicious, giving us an almost equal increase. There are 279 rules (less than 1% of the rules) that have a `detection_filter` option.

The `ip_proto` option checks the protocol number on the IP address. This number is used to determine the protocol on the next level of the network. The increase in detection is twice as large for malicious traffic as it was for benign traffic. While removing the protocol number picked up some random traffic, it is possible the removal identified variants of the malicious traffic that used new protocol numbers. Only 33 rules have the `ip_proto` option.

The option `dce_iface` checks DCERPC (distributed computing environment remote procedure calls) interface. The `dce_iface` option is designed to normalize the universally unique identifier that clients use to communicate with servers under this protocol. The Snort 3 documentation describes this option as a means to eliminate false positives from multiple services being connected to the server [8]. It then follows that removing this option increases false positives at a greater rate than true positives, giving us a bad classifier for single removals. Only 235 rules have the `dce_iface` option.

The combinations that form the last Pareto frontier are heavily influenced by the single options within each group. The jump in false positives seen in `dce_iface` and `flow` comes from the removal of `dce_iface`. The removal of content, `ip_proto`, and `isdataat` is likely a random increase, rather than a useful modification. **Summary:**

- The changes caused by removing `detection_filter`, `flags`, and `http_header` are random.
- Removing `dce_iface` results in a much higher rate of false positives.
- The most effective modification is removing content from multi-content rules, due to the increased sensitivity to variants to exploits.
- The options `flow` and `ip_proto` only appear on the Pareto frontier when paired with other removals.

h) Effective Modifications: Based on the macro f1-scores, removing content from multi-content rules is the most effective modification. With the size of this subset, we find that 57% of the rules are candidates for effective modifications. For achieving stronger defense at lowest cost to false positives, removing a single content from multi-content rules is the most effective modification. We see this demonstrated not only in the macro f1-scores, but also from the utility function. For reducing false positives at the cost of detection, the ideal modifications are inverted results of removing `flow` and `dce_iface`.

V. CONCLUSION

Characterizing the performance space of SIDS enables novel approaches to studying network traffic. Through modifications, SIDS rules can be used to collect various sets of traffic based

on the desired outcome. We demonstrated that through the most effective removal of a single content option from rules with at least two content options, the performance space expands by 11.3%. We distinguished the difference between effective modifications and modifications that result in random changes detection. Through these modifications, SIDS rules can be used for other purposes than just network intrusion detection systems.

ACKNOWLEDGMENTS

This material is based upon work supported by, or in part by, the National Science Foundation under Grant No. CNS-1805310, and the U.S. Army Research Laboratory and the U.S. Army Research Office under Grant No. W911NF-13-2-0045. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

REFERENCES

- [1] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, Jan. 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804512001944>
- [2] M. Roesch, “Snort – Lightweight Intrusion Detection for Networks,” 1999.
- [3] “Snort - Network Intrusion Detection & Prevention System.” [Online]. Available: <https://snort.org/>
- [4] C.-Y. Ho, Y.-C. Lai, I.-W. Chen, F.-Y. Wang, and W.-H. Tai, “Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems,” *IEEE Communications Magazine*, vol. 50, no. 3, pp. 146–154, Mar. 2012.
- [5] G. Tjhai, M. Papadaki, S. Furnell, and N. Clarke, “Investigating the problem of IDS false alarms: An experimental study using Snort,” in *Proceedings of The Ifip Tc 11 23rd International Information Security Conference*, S. Jajodia, P. Samarati, and S. Cimato, Eds. Boston, MA: Springer US, 2008, vol. 278, pp. 253–267. [Online]. Available: http://link.springer.com/10.1007/978-0-387-09699-5_17
- [6] E. Goodarzi, M. Ziae, and E. Z. Hosseini, *Introduction to Optimization Analysis in Hydrosystem Engineering*. Springer Science & Business Media, Feb. 2014.
- [7] E. Pauley, P. Barford, and P. McDaniel, “DScope: A Cloud-Native Internet Telescope,” in *Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023.
- [8] M. Roesch, *Snort 3 User Manual*. [Online]. Available: <https://usermanual.wiki/Document/snortmanual.1752822391.pdf>
- [9] R. L. Graham, “An efficient algorith for determining the convex hull of a finite planar set,” *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, Jun. 1972. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0020019072900452>
- [10] P. Richter and A. Berger, “Scanning the Scanners: Sensing the Internet from a Massively Distributed Network Telescope,” in *Proceedings of the Internet Measurement Conference*, ser. IMC ’19. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 144–157. [Online]. Available: <https://doi.org/10.1145/3355369.3355595>
- [11] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, “Network Telescopes: Technical Report,” p. 14.
- [12] C. Tsaoousis, “firehol_level4 by FireHOL, attacks IPs list, at FireHOL IP Lists.” [Online]. Available: http://iplists.firehol.org/?ipset=firehol_level4
- [13] D. Powers, “Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation,” p. 24.
- [14] Y. Sasaki, “The truth of the F-measure,” p. 5.
- [15] J. Opitz and S. Burst, “Macro F1 and Macro F1,” Feb. 2021. [Online]. Available: <http://arxiv.org/abs/1911.03347>
- [16] U. Aickelin, J. Twycross, and T. Hesketh-Roberts, “Rule Generalisation in Intrusion Detection Systems using Snort,” *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 1, p. 101, 2007. [Online]. Available: <http://arxiv.org/abs/0803.2973>