Protecting Run-time Filters for Network Intrusion Detection Systems

Victor C. Valgenti  
Petabi, Inc.  
Irvine, California 92612 U.S.A.  
Email: vvalgenti@petabi.com

Hai Sun  
Department of EECS  
Washington State University  
Pullman, WA 99163 U.S.A.  
Email: hsun1@eecs.wsu.edu

Min Sik Kim  
Petabi, Inc.  
Irvine, California 92612 U.S.A.  
Email: msk@petabi.com

Abstract— Network Intrusion Detection Systems (NIDS) examine millions of network packets searching for malicious traffic. Multi-gigabit line-speeds combined with growing databases of rules lead to dropped packets as the load exceeds the capacity of the device. Several areas of research have attempted to mitigate this problem through improving packet inspection efficiency, increasing resources, or reducing the examined population. A popular method for reducing the population examined is to employ run-time filters that can provide a quick check to determine that a given network packet cannot match a particular rule set. While this technique is an excellent method for reducing the population under examination, rogue elements can trivially bypass such filters with specially crafted packets and render the run-time filters effectively useless. Since the filtering comes at the cost of extra processing a filtering solution could actually perform worse than a non-filtered solution under such pandemic circumstances. To defend against such attacks, it is necessary to consider run-time filters as an independent anomaly detector capable of detecting attacks against itself. Such anomaly detection, together with judicious rate-limiting of traffic forwarded to full packet inspection, allows the detection, logging, and mitigation of attacks targeted at the filters while maintaining the overall improvements in NIDS performance garnered from using run-time filters.

Keywords—Security; Network Security; Filters; Run-time Filters; IDS; Intrusion Detection; Deep Packet Inspection;

I. INTRODUCTION

Network Intrusion Detection Systems (NIDS) examine a massive number of network packets in the quest to identify the few packets that are known suspicious. The large number of rules defining these potentially malicious packets coupled with the fact that most packets are benign leads to much wasted effort. As illustrated in TABLE I malicious traffic accounted for less than .0828% of the total traffic in relatively attack-rich packet captures. Run-time filters can serve to greatly reduce this wasted effort by quickly filtering out packets that have no chance of intersecting with the rules. Such filtering helps the NIDS devote most of its effort to examining those packets most likely to contain malicious traffic. Thus, there is strong motivation to employ run-time filters. Unfortunately, as demonstrated in our previous work [19], it is trivial for an attacker to craft traffic that simply mimics malicious traffic and causes the run-time filters to examine every packet. Since the run-time filter comes with some added overhead it is possible that a NIDS with run-time filters could actually perform worse than a NIDS without any filtering when under such pandemic conditions.

The key to defending against this weakness is to accept run-time filters as an orthogonal detector to the NIDS normal packet inspection. This simplified detector can safeguard the purpose of the run-time filter by limiting the flow of packets forwarded to the NIDS while monitoring for anomalies directed at the filter. In this manner, it is possible to both protect the value of the run-time filters in the general case (i.e. less traffic forwarded to the NIDS for full inspection) while maintaining a complimentary audit trail for detecting and identifying attempts to circumvent or exploit those filters.

II. RESEARCH IN FILTERING SOLUTIONS

Researchers have developed several run-time filtering systems. Sachidananda et al. [12] create a scheduling scheme that attempts to prioritize likely malicious packets. Thus, as the NIDS becomes burdened, those packets deemed more likely to be malicious are first to get resources and less likely to be dropped due to resource exhaustion. The choice of malicious candidates is based on run-time filters that use pre-configured blacklists as well as simple string matching tokens. Other research has sought to create hardware devices that can match simplified versions of a rule set of know malicious patterns, forwarding to the NIDS only that traffic that might match some rules used by the NIDS [1], [14], [17], [18]. The parallelism available in specialized hardware like Field Programmable Gate Arrays (FPGA) can allow for many simple pattern matchers to act in unison, determining if any potentially malicious strings exist in the packet, and then
collating the results. Only traffic intersecting with the rule set is forwarded to full NIDS processing; typically between 10–20% of the total traffic [1], [14], [18]. Even the Open Source NIDS Snort [15] employs run-time filtering. It does a one-time match of each packet using a fast-pattern matching algorithm to identify packets that might match rules so as to avoid exhaustive deep-pattern matching for those that cannot match any rules.

Another tactic is to employ statistical sampling by valuing packets occurring in specific locations within network flows (traffic between two endpoints with fixed port numbers) and only considering some of the packets occurring elsewhere in the flow [11]. Under this paradigm, it was found that most malicious traffic is found early in a flow, a fact corroborated in TABLE I where more than 92% of all detected malicious traffic occurs within the first five packets of a given network flow. Limmer et al. [7] illustrated a slightly more advanced technique by prioritizing packets at dialectal shifts in a network flow. In other words, preference is granted to network packets near changes in the overall direction of data in network flows. This helps to limit the exhaustive examination of large data transfers.

III. ATTACKING RUN-TIME FILTERS

Filtering systems, as discussed in this paper, imply systems that use a simplified fixed-string matching automata to determine that a packet cannot match any rule in a rule set. We acknowledge that sampling, and filtering, when employed with anomaly detectors beyond fixed string matching may have detrimental effects on the anomaly detector [2], [9], [10] stemming from the fact that anomaly detectors typically depend on large trends in data not necessarily visible when examining a subset of the total population.

The benefit of filtering stems from the fact that the filter removes enough traffic from full evaluation to amortize the cost of the filter and ultimately require less resources overall. Since NIDS typically have constrained resources, the goal of any run-time filter is to ensure that those resources are applied to the most likely candidates. The effort, or work, of a NIDS employing run-time filters is thus a simple accounting function:

$$\text{work} = \alpha \lambda + p \beta \lambda .$$

In this equation, $\alpha$ represents the mean amount of time required by the run-time filter, $\beta$ the mean time required by the NIDS for full inspection, and $p$ represents the general frequency, or probability, of packets forwarded to the NIDS with $\lambda$ representing the mean rate of traffic. Given this equation it is possible to estimate the amount of work required for the NIDS and filter together to process some $\lambda$ number of packets. As such, the effectiveness of the run-time filter becomes a function of how many packets are removed from full inspection. The Filter Efficiency (FEF) is defined as:

$$\text{FEF} = \frac{\beta}{\alpha + p \beta}$$

which simplifies and normalizes the work equation such that an FEF greater than 1 represents a filter that is reducing the overall system burden while an FEF less than one represents a system that is actually performing worse than if no filters were involved.

Essentially, any packet forwarded to the NIDS must first pass through the run-time filter. This increases the cost for those packets. Packets not forwarded to the NIDS, however, do not incur the added overhead of a full evaluation. As long as $\alpha$ remains smaller than $\beta$ and/or $p$ remains small enough, then the run-time filter can offer much improvement. Fig. 1 illustrates the FEF for a ratio of $\alpha$ to $\beta$ between $\frac{1}{10}$ to 1 and $p$ values from 0 to 1. As can be seen the ratio of $\alpha$ to $\beta$ greatly widens the gap between top performance and worst case. However, as the value of $p$ increases the effectiveness of the filters rapidly falls to 1, or worse. This demonstrates the expectation that run-time filters must operate at less cost than the full NIDS inspection as well as reduce the number of packets sent to full evaluation in order for those filters to provide value.

Most run-time filters work by using some piece of each string pattern found in the rule-set [1], [14], [15], [17], [18]. A small automata is made from these partial strings, potentially combined with other information like destination ports or IP Addresses. The run-time filter then matches all incoming traffic against this simplified automata, which we assume can operate at line speeds. Any packet containing a match with this small automata may be malicious. However, any packet not matching this automata is guaranteed not to match any rule in the rule-set. Thus, those packets can be safely ignored. In the average case, this works quite well since the probability of normal traffic matching the rule set tends to be quite low, though poorly crafted rules are an exception to this and not covered in this work. In a worst case scenario, as illustrated by a ratio of $\alpha$ to $\beta$ equal to one in Fig. 1, the NIDS and run-time filter together may operate at higher cost in resources than a NIDS without any filtering.

![Illustration of Filter Efficiency](image-url)
The problem is that an attacker can trivially create a worst-case scenario. It is quite easy to generate traffic that will match patterns in a rule set as we illustrated in previous work [19]. Thus, it becomes possible to craft bursts of packets that will pass through the run-time filter. Even worse, these packets can be crafted to exert heavier burden on the NIDS without causing any alerts that might warn the system stakeholders of the attack; similar in spirit to the algorithmic complexity attack [13]. Assuming the NIDS is incapable of handling line-speed traffic, which is often the case, the NIDS will begin dropping packets from evaluation as system resources are exhausted. Since the attacker can control, to a large degree, the traffic that the run-time filters and NIDS will see, the attacker can create a burst of traffic that will exhaust NIDS resources in the hopes of hiding an attack. In our own laboratory, we created attacks as illustrated in Fig. 2, where we crafted a set of 100 packets capable of bypassing filters by partially matching a rule in the target NIDS rules database. We then sandwiched a set of five attack packets between two sets of the partially matching traffic. We then generated traffic at maximum send rate on a Fast Ethernet connection (100 Mbps) directly at the target NIDS (Snort in this case). Since the partially matching traffic contained signatures almost matching rules, all the crafted traffic was passed to the NIDS along with the malicious traffic. This overloaded the NIDS, and resulted in only one of the five attack packets acknowledged by the NIDS and none of the partially matching traffic (as expected). We modified the attack so that it contained only 3 attack packets and were able to sneak all three attack packets past the NIDS with it never registering a single attack for every time we ran the test. Essentially, this attack exhausts the NIDS resources before the actual attack, then continues to exhaust the NIDS resources to make certain it does not recover until the attack has passed. In this manner, it is possible to sneak an attack past the NIDS. Worse, only examining the number of packets dropped by the OS offers any clue that something suspicious has happened.

IV. PROTECTING RUN-TIME FILTERS

As was illustrated in Fig. 1, the benefit of the run-time filter remains only so long as the Filter Efficiency (FEF) remains greater than 1 (see Section III). In fact, the filter acts as a rate limiter for the NIDS as well as a means of judgmental sampling—only packets that might match are forwarded. As such, protecting run-time filters requires protecting two aspects of the run-time filter. First, the rate-limiting capabilities of the run-time filter must be maintained even in the face of pandemic conditions. Second, security must be maintained in the face of evasive tactics by an attacker. In all, it is as depicted in Fig. 3.

A. Rate Limiting Run-time Filters

Rate limiting for run-time filters requires limiting the number of packets that are forwarded to the NIDS for full evaluation. This has the benefit that the rate limiter can ensure that the NIDS is never, or almost never, overwhelmed. However, this means that when under pandemic conditions, the rate limiter must remove some traffic from full deep-packet inspection. Any traffic that has passed through the run-time filter and has matched one of the fixed strings in the run-time filter has the potential to contain malicious traffic. Further, the forwarding mechanism must employ a means of deciding what packets to forward that cannot be predicted by a potential attacker. To accomplish this we employ the tactics as illustrated in Random Early Drop [5] and the Token Bucket (Leaky Bucket) theory.

First, we recognize that for any particular NIDS a fixed number of packets can be accommodated at any interval, and that packets will be evaluated at some rate. Further, we must recognize that if the NIDS is overwhelmed packets will be dropped from evaluation. Even worse, such packet drop can be controlled from an external source (i.e. a hacker transmitting a stream of partially matching packets). Regardless, considering these factors it is possible to determine some value, \( \gamma \), that represents the maximum burst of traffic that a particular NIDS can handle (i.e. the NIDS buffer). Further, it is possible to estimate a value, \( \delta \), which represents the average number of packets processed in some fixed time-period \( \tau \) (some fraction of a second). We use \( q \) to denote the number of packets currently forwarded to the NIDS for any time period (i.e. number of packets in the queue). Thus, like a leaky bucket, it would be possible to accept a burst of packets up to size of \( \gamma \) (essentially \( q = \gamma \)) and then not forward any packets until the current time period \( \tau \) is finished. At the start of the new time period \( \tau \), \( q \) is reduced by \( \delta \), and \( \delta \) more packets may be forwarded to the NIDS if more packets are flagged in the new time period. This serves to rate-limit the number of packets forwarded to the NIDS for any time period \( \tau \) to no more than \( \gamma \) packets which should be set to a value which the NIDS can
consume without dropping any packets. This would ensure that any packet sent to the NIDS would not be dropped due to resource exhaustion and thus could not bypass the system silently as illustrated in Section III.

Unfortunately, all flagged packets in some time period $\tau$ after the reception of $\gamma$ packets would be dropped. Thus, an attacker need only send more than $\gamma$ packets in a single time period $\tau$, in order to bypass the NIDS, similar to the method described in Section III. To mitigate such a situation we apply the idea of Random Early Drop. When the number of packets that have been forwarded to the NIDS reaches $\gamma$ then we can begin to forward packets based on a forwarding probability as illustrated in (3), where $q$ represents the current number of packets forwarded to the NIDS. At the start of each time interval, we simply reduce $q$ by $\delta$. Since the chance of any packet being forwarded to the NIDS is random, with the lowest chance $\frac{1}{\gamma}$, an attacker cannot determine precisely what packets will pass a system. This removes much of the allure of the attack as illustrated in Section III as determining the exact boundaries of where to best hide malicious traffic among specially crafted packets is now just a best guess.

\[
\text{ForwardingProbability} = \begin{cases} 
\frac{\gamma}{q} & \text{if } q < \gamma \\
\frac{1}{\gamma} & \text{if } q \geq \gamma
\end{cases}
\] (3)

Despite the resilience this provides the filters, this still suffers from two significant issues. First, potential malicious traffic may not be forwarded to the NIDS and thus might not be evaluated. An attacker can increase the likelihood of this event by increasing the number of crafted “pseudo” malicious packets. Of course, the attacker cannot dictate that a malicious packet will bypass the system, but can play the odds to make it happen. Secondly, the random nature of forwarding of packets means that more packets than the NIDS can handle may be forwarded to the NIDS. The latter problem can be mitigated in system by crafting values for $\gamma$, $\delta$, and $\tau$ that fit a worst-case scenario for the physical system. The former problem, however requires the use of an anomaly detector.

B. Turning Run-time Filters into an Anomaly Detector

The primary problem with run-time filtering is that it is difficult to determine when the filters are under attack, and thus indirectly the NIDS. In the average case, we can predict that it is unlikely that many packets will match rules in the rule set. Not only is the typical amount of malicious traffic in a network tiny compared to the total volume, it is very unlikely for normal traffic to match a set of well-tested rules (poorly crafted rules a definite exception to the prior statement). In Section IV-A we set $\frac{1}{\gamma}$ as the point to begin limiting the flow of packets to the NIDS. This value was chosen for several reasons. First, it allows up to $\frac{1}{\gamma}$ packets flagged by the run-time filter to pass directly to the NIDS. Real network attacks tend to be quite bursty in that a series of packets that match rules in a NIDS rule set will follow each other in a rapid series, often because the NIDS is flagging aspects specific to a particular flow of Internet traffic. Further, we recognize that most network traffic is comprised of relatively short-lived flows [6], and that packets that might match a rule due to a particular flow are likely to match less than 10 packets as most flows are 10 packets or less. Thus, if $\gamma$ is set to 20 or higher then it can be expected that a typical burst of malicious traffic from a single flow can be forwarded to the NIDS without any packets filtered from full evaluation. Further, in our evaluations of alerts we note that more than 80% of the alerts generated from the traffic in TABLE I apply to the same event. In other words, the NIDS generates a large number of redundant alerts. Thus, by ensuring the capture of $\frac{1}{\gamma}$ packets in a burst of packets matching a filter, it typically ensures that we will capture the actual event causing the alerts even if we do not get all of the packets from that event. In other words, since attack events typically create a large volume of temporally co-located packets, all of which will set off alerts in the NIDS, capturing the first couple packets should suffice to identify the event causing the alerts. Finally, we note that a $\gamma$ of 20 is quite low and expect that the value for $\gamma$ would be much greater. However, the exact value for $\gamma$ depends on the system in question and can only be determined through profiling the NIDS.

During normal traffic, if the boundary of $\frac{1}{\gamma}$ is exceeded, an alert is posted by the run-time filter. In this manner, traffic that might bypass the NIDS due to the NIDS being overburdened will now have an audit entry from the run-time filter. The information provided will not have the level of detail that a full alert may have, but will provide a list of rules that the packet may have matched. This is desirable even if the packets are benign as it can serve to identify overly broad run-time filters and/or rules that might need further refinement. Further, if the NIDS is attacked, it provides an audit trail that will not only warn of the anomaly, but also provide some means for narrowing the culprits. Finally, keep in mind that when under attack as illustrated in Section III that packets that are noted by the filter here but not forwarded to the NIDS may completely skip NIDS evaluation due to resource exhaustion if the filter protection were not in place. Thus, while this system seems to be sacrificing information, it is actually preserving the audit trail that would normally be lost when a system becomes overburdened.

The information captured by the filter is as follows. First, for every packet that would have been forwarded by the run-time filter, but is dropped instead, it is possible to create a list of possible rules that packet might have matched based on the run-time filters matched. Thus, even if a packet is not forwarded to the NIDS, a list of potential match candidates is retained (as a given filter may apply to more than one rule). In this manner it is possible to ensure that at least some information is retained for every potentially malicious packet thus reducing the effect of the attack as outlined in Section III. Further, the value for $q$ is captured, detailing just how deep this burst of matching packets runs. In addition to that, the count of the number of purposely dropped packets is also maintained, as well as the total number of alerts. These values can be matched up with the NIDS to see if the tallies are correct or
if something has slipped past the system.

Finally, we note that a run-time filter is typically resource constrained. Thus, this solution must work with a very small footprint. This can be accomplished by maintaining a single bitmap for all possible rule matches. Thus, even a rule set of thousands of rules can be reduced to only hundreds of bytes. The bitmap can be indexed by rule ID and thus quickly accessed. When a packet is dropped, all rules that packet might have matched are set to 1. This implies a loss of information over time. However, we note two aspects that mitigate any real loss of data. First, the data in the bitmap must be periodically retrieved by the NIDS, and the bitmap cleared. This allows event data to aggregate to the nearest period, with that period being anything from a few seconds to a minute. Secondly, as we have noted before, typical attacks produce large numbers of redundant alerts for a single event. These alerts are tightly coupled in time. The bitmap would only ignore redundant (non-new) data. Essentially, at this level of aggregation all of the redundancy is eliminated. Performing this process once every few seconds, or up to a minute, helps to amortize the high-cost of this operation. Ultimately, given the data from the bitmap it becomes possible to ascertain the list of potential patterns engendered some interesting results. First, we began by looking for patterns in Snort, given the above rule set, and then used the entire Defcon 11 and 17 Capture the Flag data sets. We set $\gamma$ to a very conservative value of 25. The reason for this low value was that originally we intended to examine performance across a series of values for $\gamma$. However, as is illustrated in Subsection V-B we found that even small values for $\gamma$ were typically sufficient in the expected case. We adopted a very conservative value of 100 microseconds as the maximum time to match rules for the NIDS and a time period of .001 seconds thus making $\delta = 10$. This value came from examining the worst performing rules for Snort, given the above rule set, when under operation. The goal of our evaluation is to determine how well this technique might fit actual operation.

A. Efficiency of Protecting Run-time Filters

As was pointed out in Section III, in order for the run-time filters to remain valid they must reduce the amount of traffic forwarded to the NIDS, as well as provide minimal processing time compared to the NIDS. In Section IV-A we discussed techniques that might be used to reduce the cost of these additions to the run-time filters. First, we considered common rule sets for NIDS. Since 2010, the Sourcefire Vulnerability Research Team’s (VRT) Snort rule set has fluctuated between 5,000 and 3,000 default enabled rules. As such, the largest component of this solution, the bitmap of possible match rules, would require at most 625 bytes to implement. Even allowing for up to 8000 rules, only 1 KB of memory would be required for this solution. Beyond this, the only other memory overhead would be counters for current number of packets sent to the NIDS, Number of packets dropped, and number of packets counted as anomalies. Thus, in terms of memory, this solution can be accomplished in roughly 1 KB of memory or less; a relatively small amount even in resource-constrained systems.

This solution would also add some extra processing for each packet inspected. For every packet that potentially matched a rule, this extra processing would be maintaining the counter for the current queue size, a tiny addition percentage-wise compared to navigating the matching automata. However, there are a couple processes requiring more effort. First, when an alert packet is dropped, all of the indices for possible match rules must be updated in the bitmap. Theoretically, this could be as many as $m$ rules, where $m$ represents all the rules in the rule set. While that instance would greatly increase processing, it is largely a theoretical problem for the following reasons. First, the filters are culled from all rules. If a single filter works for two rules, then only one filter is included in the run-time filter. Thus, for a packet to potentially match all rules, then there would only be one filter to match against. In fact, in the VRT rule set used in our tests, less than 2% of the filters pointed to more than 5 rules, with the largest filter pointing to only 66 rules. Further, the cost for adding these rules to the bitmap is greatly amortized by the extremely small number of packets that are ultimately dropped. In our examinations, illustrated in Subsection V-B, out of millions of packets only a tiny number of events actually occur (less than .01%) requiring the bitmap to be used. Thus, we believe this will not impede the system under expected conditions, though under pandemic conditions the cost for the filter may increase. The rate-limitier, however, enforces the rate of packets forwarded to the NIDS ensuring that the NIDS can consume all, or nearly all, the packets forwarded for full deep packet inspection. Thus, the value for $p$, as discussed in Section III, would remain low preserving the PEF of the filters.

B. Run-time Filters and Traffic

Examining our algorithm when applied to actual traffic patterns engendered some interesting results. First, we began with a value for $\gamma$ of 25. This is an extremely small value for representing the buffer of a NIDS, but we adopted it to see the results when under such a small buffer. What we found was that in normal traffic, even traffic with lots of attacks in it, attacks rarely occurred in bursts larger than 10 packets. Thus, the algorithm rarely witnessed any “anomalies” due to $q$ growing larger than $\frac{2}{\gamma}$. Fig. 4 illustrates these trends. First,
the DARPA data showed no anomalies as the attack traffic is sparse and the network speed relatively slow. The Defcon data sets, while showing some few anomalies, suffered from the same problem. The WSU data, however, was collected from a gigabit link on a university campus. The higher rate of data was fundamental in creating the larger number of events. Even so, this chart also illustrates that most of the events occurred with \( q \) overflowing the \( \frac{1}{2} \) watermark by only a few packets for only a brief time. Increasing the size of \( \gamma \) quickly reduced the number of anomalies to a handful as is represented by the datasets with a number greater than 25 after the name in Fig. 4. In this manner, it would be possible to tune a system to the expected traffic of a network such that anomalies are quite rare. This increases the value of the anomalies as they stand out and can garner more attention from security staff. Ultimately, we note that in the average case the number of events is quite small, and the average size of \( q \) is quite small, as illustrated in Fig. 5. Essentially the NIDS buffers will fill up sometimes (as \( q \) grows), but mostly will empty very quickly (as \( q \) is reduced back to 0).

C. Maintaining Security

While the rate-limiter serves to maintain the value of the run-time filter, it is probabilistic and thus there still is a small chance that a packet forwarded to the NIDS is not inspected due to resource exhaustion. As a simple example, we pushed 1250 100 byte packets, each designed to match a rule, through our simulator in a single time period of .001 seconds (slightly more than the maximum number of packets possible on a Gigabit link in that time frame). For this test, we also increase \( \gamma \) to a more reasonable size of 100, but otherwise left all other variables the same. Under such pandemic conditions only 57 more packets than intended were passed to the NIDS which could result in dropped packets by the NIDS. Of course, filling a gigabit link with this much traffic is not a trivial undertaking and would require a distributed attack system or a hardware traffic generator. Even then, the generated traffic would likely compete for bandwidth with other flows. Further, it would become very difficult for the attacker to actually arrange the traffic to reliably repeat the attack as outlined in Section III. Regardless, this hole can be plugged by simply tracking every packet in the bitmap once the \( \frac{1}{2} \) watermark is exceeded. While this would incur more overhead the statistics in Subsection V-B show it is possible to tune \( \gamma \) such that normal traffic will not exceed the \( \frac{1}{2} \) watermark and thus this extra overhead will be greatly amortized.

For a final test, we recreated the attack scenario as outlined in Section III. We ran the test against the simulator 100 times for \( \gamma \) at 25, 50, and 100. In all cases, The filter system registered the attack packets, as expected since it is registering all packets potentially forwarded to the NIDS (once the \( \frac{1}{2} \) watermark is exceeded). Further, in the worst case, with a \( \gamma \) of 25, only around 15 packets more than the expected load for the time period were forwarded to the the NIDS. For a \( \gamma \) of 100 that number ranked slightly more than 2. While it is possible that those extra packets might arrive at the NIDS while the NIDS resources are full and ultimately avoid processing, it is unlikely that an attacker can manipulate the traffic enough to reliably move any specific packet past a NIDS without detection. While this system may not be perfect, it ensures an audit trail is maintained.

VI. DISCUSSION

Protecting run-time filters may seem a little like an impossible fight. After all, the run-time filters are already there to help protect the NIDS from too much traffic. Adding another layer to protect the filters seems to lead to another layer to protect the protection of the filters and so on. However, all the run-time filters we have examined consider only expected case traffic. While such myopic evaluation is reasonable from a functional point of view, it does not stand in the adversarial environment that is computer security. As such, run-time filters, especially those for security devices, must consider and account for potential attacks as illustrated in this paper, and in others [3], [13], which attack some part of the NIDS. Otherwise, if the filters are easily bypassed as demonstrated in Section III, then they only offer a false sense of security. Further, by turning the
run-time filter into its own detector, it is possible to leverage existing aspects of the system to provide a deeper security. It is this deeper security that provides the compensating control to the run-time filter and ensures an audit trail when attackers attempt to abuse or bypass the filters. As such, another layer has not been added so much as the system is, as a whole, more fully utilized.

We have not covered every possible detail an attacker might use to abuse the run-time filter or this protection. For example, an attacker could use fragmented packets to attempt to push the algorithm into its highest drop probability. However, we note that for true filtering, the run-time filter itself will need to re-assemble packets, thus eliminating this issue. Even if that were not the case, it would be possible to base forwarding on the number of bytes seen rather than packets [11]. Regardless, we do not consider every possibility in this paper. However, we do address the primary issue, and that is the enforcement of the value of the filter through rate-limiting while maintaining an audit trail to preserve security.

VII. CONCLUSION

We have proposed a method for protecting run-time filters from attackers. Not only does this technique greatly mitigate the risk to run-time filters from attackers, but it also provides a complementary anomaly detector that can be used to detect large bursts of traffic that intersect with a NIDS rule set. This is desirable not only to detect potential attackers, but to also identify traffic trends which might imply the NIDS rule set needs refinement. Further, this new detector can be worked into larger, distributed, NIDS to provide another data point capable of enhancing the overall understanding of the system. Ultimately, as bandwidth increases traffic will also increase. It is our firm belief that naively examining every packet as a means of security will grow less and less effective. Run-time filters offer great value to help target NIDS to the most suspect elements of traffic, and the techniques as outlined here can mitigate the risks associated with those run-time filters.

REFERENCES


