

The Comparison of IP Networks

S Terry Brugger*

December 21, 2005

Abstract

One of the many criticisms of the DARPA IDS evaluation is that it did not evaluate traditional, signature based, off-the-shelf intrusion detection systems. We performed such an evaluation on the 1998 dataset using Snort to determine the usefulness of the DARPA dataset, and found that overall detection performance was low and false positive rates were unacceptable. A more detailed analysis showed that the dataset did a good job of modeling attacks that are difficult for signature based detectors to find; however, we found no evidence to support that the false positive rates – which result from the “normal” data in the dataset – were in any way indicative of real world networks. In order to validate future “normal” data, we propose to develop a methodology to quantify the similarity of two network traces. Such a methodology should be useful for comparing real and artificial network traces, as well as different real networks, and the same network over time.

1 Introduction

In 1998 DARPA recognized the need to be able to perform quantitative evaluations on intrusion detection systems. MIT’s Lincoln Labs was contracted to work with the Air Force Research Laboratory in Rome, NY to build an evaluation dataset and perform an evaluation of the then current IDS research being funded by DARPA. They built a small network intending to simulate an Air Force base connected to the Internet, producing background activity with scripts, and injecting attacks at well defined points, and gathered tcpdump, Sun BSM, process and filesystem information. The DARPA funded IDS researchers were given seven weeks of this data with a list of when and where the attacks occurred, to train their systems. Once they trained their systems, two weeks of data were used to test the detection performance of the intrusion detection systems. The results of this evaluation were published in (Lippmann, Fried, Graf, Haines, Kendall, McClung, Weber, Webster, Wyschogrod, Cunningham, and Zissman 2000; Lippmann, Haines, Fried, Korba, and Das 2000).

*Department of Computer Science, University of California, Davis, One Shields Ave, Davis CA 95616, <zow@acm.org>, WWW home page: <http://bruggerink.com/~zow>

After the evaluation, the researchers whose systems were evaluated, and others in the community provided feedback on the evaluation. This resulted in changes for the 1999 evaluation, such as using more stealthy attacks, including a Windows NT target (and its audit logs), defining a security policy for the target network, and testing more recent attacks.

While the 1999 evaluation addressed some of the concerns voiced in the IDS research community, there remained serious reservations regarding its applicability. McHugh published an in-depth criticism of the evaluation based solely on the procedures used in building the dataset and performing the evaluation (McHugh 2000). While Lincoln Labs made the datasets available after the evaluations and many researchers used this data as the basis of evaluating their systems, after McHugh’s criticism was published, the use of these datasets was more closely scrutinized by reviewers. In 2003 Malhony and Chan decided to look more closely at the data itself. They discovered that the data included irregularities, such as differences in the TTL¹ for attacks versus normal traffic, that even a simplistic IDS could identify and achieve better performance than would ever be achieved in the real world (Mahoney and Chan 2003). Unfortunately, since the publishing of these criticisms, and possibly because of them, new efforts to provide datasets for IDS researchers, such as PREDICT (RTI International 2005) focus on the use of real – not synthetic – traffic. The problem with real traffic is that we lack ground truth – we can not positively label many connections as malicious or benign. For example, a TCP/IP packet arriving on port 80 to a machine which does not have a web server may just be a user accidentally typing an address, or it may be a probe used as a precursor to an attack.

McHugh’s primary criticism of the evaluation was the failure to verify that the network realistically simulated a real-world network. Malhony and Chan’s work proved that, in fact, the data did not simulate real world conditions, in certain important aspects. Another criticism made by McHugh was that, while the evaluation was performed to test the performance of advanced intrusion detection systems, traditional, signature-based IDSs were not run against the data to see how they would fare. If existing IDSs performed satisfactorily, either more advanced systems are not necessary, or the dataset does not model sufficiently advanced attacks that require more advanced systems to detect. Such an evaluation should serve as a baseline against which all other systems are evaluated. Lincoln Labs provided such a baseline though the use of a naive keyword-based detector, which they claimed had a 30% intrusion detection rate for anywhere from zero to 100% false positives. In fact, Lippmann et. al. published a paper on how the performance of this keyword detector could be improved (Lippmann and Cunningham 1999).

Given Chan and Malhony’s work, it appeared that if an advanced IDS could not perform well on the DARPA dataset, it would not perform acceptably on realistic data. This property suggested that the DARPA dataset may still be useful for a first-order IDS evaluation. In order to test this hypothesis, we generated

¹IP packet Time To Live

a detection baseline of the dataset with a contemporary version of Snort, the open-source, signature-based intrusion detection system (Caswell and Roesch 2004). We chose Snort as it is readily available to anyone who wishes to validate our results². While other off-the-shelf ID systems may provide incremental improvements over Snort in certain areas, our interest is in the general behavior of the DARPA dataset, not in performing an intrusion detector shootout. We expected that given the six years between the dataset creation and the Snort ruleset that we are using, that we could achieve excellent detection performance and could minimize the false positive rate with some fine tuning. Put another way, given the time span between the dataset and the ruleset, we expect that any attacks in the dataset that could be detected by a signature based system would be detected by the ruleset we used.

Pickering performed a similar evaluation for his undergraduate thesis, which we discovered only after we concluded our study. After reviewing his work, we found that his premise was different from ours, which affected his conclusions. Further, he considered the fact that Snort could achieve better performance by tuning it to the dataset, to be a failing of the dataset; an opinion we do not share. Finally, he doesn't correlate the alert to the attack, which we found demonstrates that most of the attacks are only detected through the generation of some artifact that is atypical on most networks (Pickering 2002). What we found instead, though, was that detection performance was very low, even as the false positive rate soared past the 50% guess rate line. We initially thought this may be due to a failure in the DARPA dataset to properly model attacks, or perhaps due to old rules being removed from the Snort distribution. Instead, our analysis indicates that the dataset does in fact model attacks that Snort has trouble detecting. More to the point, Snort may detect only a small portion of the connections in many types of attacks, skewing the performance figures against it. Further, many of the rules that detect these portions of attacks are policy based and produce an unacceptable rate of false positives.

This lead us to the conclusion that the DARPA IDS evaluation dataset is still useful for testing intrusion detection systems in that good performance against it is a *necessary but not sufficient* condition to demonstrating the capabilities of an advanced IDS. More importantly, we found that the network intrusion detection community is sorely lacking any means to quantitatively compare the false positive rate of intrusion detection systems. Some have speculated about the prospect of using anonymised traces of real network traffic for such a purpose; however, this would not only require a human to label the traffic as malicious or benign, but such a determination for some connections is often impossible to make on anything but the most trivial of networks.

Seemingly, the only alternative to real traffic is the generation of artificial network traffic, as was done in the DARPA evaluation. As McHugh observed, we must be able to validate that the artificial traffic is sufficiently similar to real traffic for it to be useful in the evaluation of systems. The ideal artificial

²The same argument could be made for Bro, however we have found that while Bro is useful as a research IDS, it is not commonly used in production environments, and hence its signature base is not as actively maintained.

trace would be indistinguishable from an anonymous network trace (such as those available from the Predict project (RTI International 2005)) This leaves us with one of two problems: either the evaluation data *is* an anonymous network trace (or constructed from multiple traces) – which leaves us with all the problems of real data – or we have to be able to quantify what we mean by “indistinguishable”.

One might consider diving deeper into the problem and looking at the model used to generate the synthetic data. If the model is correct, then it follows that the traces it generates should be correct. While first-order models of the network may look promising, particularly as they are easier to validate, the characteristics of modern computer networks, even small ones, are too numerous to consider every variable, and since many of the higher-order characteristics are dependent on variables that may be missing or slightly off in their measurements, the chaotic nature of the network (the so-called butterfly effect), makes the higher order values created by such models meaningless. First principle approaches are extremely useful for validating models to first order. For example, using the first-order parameter “number of hosts”, we know that the DARPA IDS evaluation data, which used a handful of hosts, does not accurately model a medium sized Air Force network, which likely has hundreds of hosts. In other words, first-principle approaches can tell us when a model is not correct; however, they can not validate the correctness of a model.

Since the first-principles approach will not serve our purposes, we must validate the models based on whether or not they can generate network traces that look sufficiently similar to real network traffic. Suddenly we are back to the need to quantify the similarity and differences between two network traces.

The driving force to date in network modeling, trace generation, and verification that those traces are accurate, has been the network engineering community. Unfortunately, this community has been primarily interested in a very limited subset of the characteristics of IP network flows, particularly the latency between packets, as this has a significant impact on the performance of networks depending on the queuing algorithm employed. There has recently been growing interest in modeling additional aspects of the network and verifying the correctness of those models; however, no evidence has been presented for the correctness of these methodologies beyond “intuition”.

We propose a new methodology for comparing network traces. This methodology is based on the use of multiple statistical measures on the set of network characteristics, and the correlations between those characteristics. For the present, we focus on the features present in IP networks and some of its higher layers such as TCP, UDP, and ICMP; however, the methodology is general and applicable to any packet network such as ATM or NetBIOS.

Such a methodology would be useful beyond the intrusion detection research community. Computer network forensics personnel could use it to compare different attack traces to ascertain the similarity between the attacks, aiding in attack attribution. Network managers could use the methodology to determine how their network is evolving over time. Network researchers might use it to measure the impact of a given protocol or application on a network. The

methodology should also be useful in showing what the similarities and differences between two different networks are, and even allow for the passive identification of the type of site the traffic comes from based only on its characteristics.

In this paper we will present our examination of the DARPA IDS evaluation dataset using Snort, starting with the methodology, followed by a detailed analysis, and concluding with our results. Then we will present the prior work that has been done in the area of network modeling and validation. We will conclude with a proposal to develop a new methodology to compare the characteristics of two network traces.

2 Evaluation of Snort on the DARPA IDS Evaluation Dataset

2.1 Methodology

Figure 1 illustrates our process to run Snort against the DARPA dataset, culminating in the generation of ROC curves (which appear in the results section).

We used the tcpdump (pcap) files in the seven weeks of training data from the 1998 data set (step 1). Each file was input into Snort version 2.1.3 (Caswell and Roesch 2004) configured with all rules and intrusion detection preprocessors enabled (step 2). An alert file was generated for each tcpdump file (step 3). Using a custom built Perl script, this alert file was matched against the list of attacks in the tcpdump file (step 4). A match was defined as the same IPs, and port numbers or ICMP types/codes. One match file was generated for each alert file (step 5). Originally we also used the timestamp; however, because the attacks are labeled by the time the malicious connection started and Snort identifies the time the packet with the attack occurs, this missed numerous matches. Not including the time corrected this problem, and while it could allow an alert and an attack to be matched erroneously, this is highly unlikely and did not occur in the large amount of data we spot checked. Further, such a case would (falsely) improve Snort's performance, and the final results would indicate that did not happen.

The matching script also reported the false negative rates per attack type (step 7) and the false positive rate (step 9) on a per rule basis. Once all the attack and alert files were matched, the lists of positive matches was filtered to eliminate duplicates, where the same rule generated multiple alerts for the same attack (step 6). Then, basic Perl scripts counted the total false negatives per attack type (step 8), and the total false positive rate per rule (step 10).

Finally, another Perl script (step 11) was used to generate the receiver operating characteristic (ROC) curves for a given set of attacks (step 12). By taking the lists of attacks correlated with Snort alerts, the script calculated the true positive rate for each rule on the given set of attacks, and divided it by the false positive rate for that rule. It then outputted the rule with the highest ratio (ordering those with zero false positives from highest to lowest true positive rate), and eliminated all attacks (and matches for those attacks) detected by that rule.

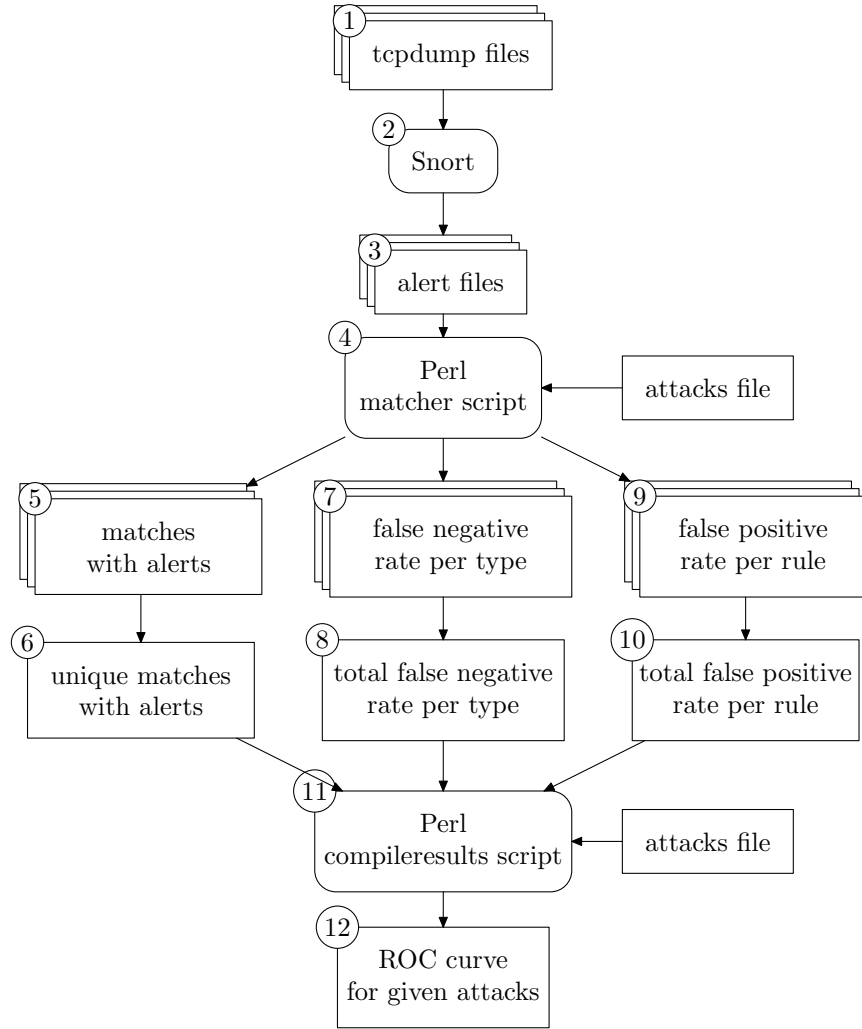


Figure 1: Basic process flow for our testing methodology. See text for detailed description.

This process was repeated until all rules were accounted for. We didn't include attacks detected by higher ranked rules in the true positive count because those detections don't provide the user with any additional benefit even though they are still saddled with the false positives that rule produces. The order that the rules are selected will follow the ROC curve from the origin (0,0) to the point at which everything is identified as an intrusion (1,1), and the selected ordering will raise the height of the curve as quickly as possible, hence maximizing the area under the curve. For example, if rule A detected attacks 1, 3, 6, 7, and 8

with 2 false positives, rule B detected attacks 1, 2, 4, and 5 with 1 false positive, and rule C detected attacks 4 and 5 with no false positives, then rule C would be output first, followed by rule A (with a 5:2 ratio beating B’s 2:1 ratio since B’s detection of 4 and 5 no longer count), and finally rule B. If a rule only detected attacks that higher ranked rules also detected, it was dropped, as it would be if we were tuning Snort for a real environment.

We plotted an overall ROC curve, inclusive of all attacks in the dataset, as well as ROC curves for the four standard categories of attacks: denial of service (DoS), probe activity, remote to local (R2L), and user to root (U2R). Since we could not find the categorization used by Lincoln Labs for which attacks were in which categories, we categorized each attack based on the descriptions provided by Lincoln Labs and our domain knowledge. We excluded the ”anomaly” attack from any of the four categories as it appears it was scored separately from the other four categories; nevertheless, it is included in the overall results. Some attacks were difficult to categorize, for example we included the ”multihop” and ”warez” attacks as R2L attacks, as root access did not appear to be necessary, and ”imap” as a U2R attack as it results in a root shell, although our domain knowledge tells us that the attack can be launched remotely.

Unfortunately, the labels for the 1998 testing data have been lost with time (Lincoln Labs no longer has them, and they were not part of the data distributed to evaluation participants), so we could not use the rule ordering generated at this stage to run against the testing data and generate the ROC curves based on the performance of the rules on the training data (essentially duplicating the process used in the evaluation). Lincoln Labs recommended using the 1999 testing data, as that addressed some of the problems with the 1998 data. Unfortunately, the format of the attack labels changed in the 1999 data, which would have meant rewriting our scripts. Given that our interest is in analyzing the dataset, and not comparing Snort to the other systems evaluated, and that none of the changes made in 1999 should have affected our general findings, we did not pursue the matter further.

2.2 Results

In this section we present our results of running Snort against the DARPA dataset. Figure 2 shows the receiver operating characteristic curves for the results overall, and for each of the four types of malicious activity. It also shows the “Random Guess” line, which represents the probabilistic performance if we guess that a given percentage of connections are malicious. Since the bulk of activity is near the origin, we use a logarithmic scale for legibility. As a side effect, the origin of the graphs is not (0,0). Also note that we use real receiver operating characteristic curves, not the “ROC” plots used by Lincoln Labs, which used a number of false positives per unit time (typically per day), instead of false positive rate, for the X-scale. Finally, we base our analysis on the number of malicious connections detected, as opposed to the number of actual attacks detected because many – especially in the DoS and Probe categories – only had a small number of their connections detected, and then only by rules

not specifically designed to detect that activity. This would be particularly problematic in an IPS (Intrusion Prevention System) which only blocks those packets seen as malicious. The following subsections will describe these cases in greater detail.

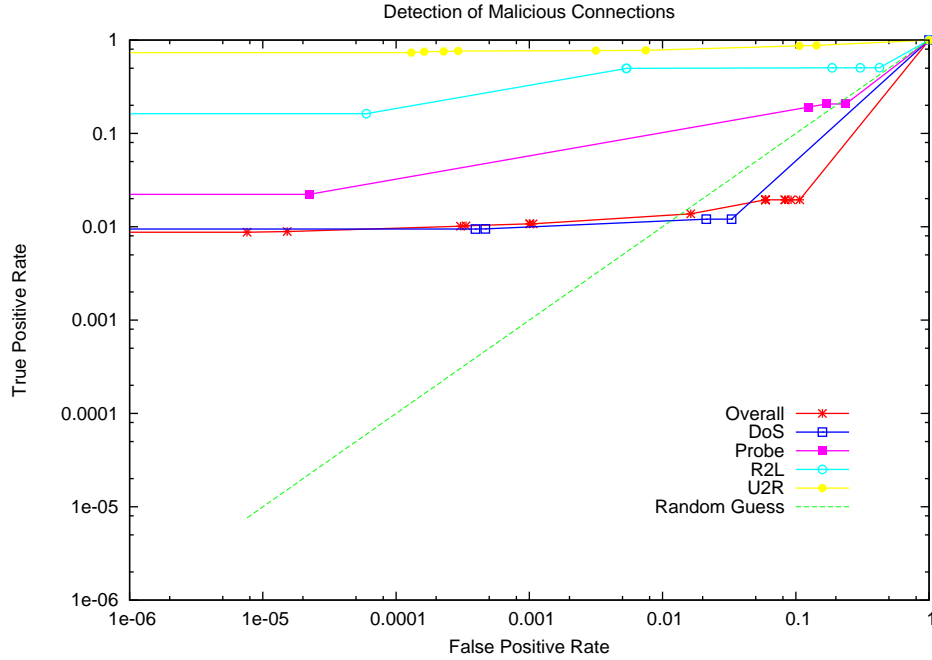


Figure 2: Receiver Operating Characteristic Curves in log,log scale. Y-axis is percentage of malicious connections detected. X-axis is percentage of alerts produced not corresponding to an attack. See text for explanation of connection versus attack detections.

2.2.1 Overall

Of the roughly 1.8 million connections containing attacks in the dataset, 1.7 million of those are from the denial of service attacks, which is to be expected given the nature of such attacks. As such, the overall results are highly skewed by Snort’s performance on the DoS attacks. We will examine the individual categories in detail below. A visual examination of the ROC curve (line with “star” points in figure 2) shows three distinct segments: those rules which have an excellent detection rate with barely any false positives resulting the initial spike, followed by a slow climb for the large number of rules that provided some detections with many false positives, and finally our extrapolation from the last

rule to (1,1).

2.2.2 Denial of Service

These days, the majority of DoS attacks are distributed denial of service attacks designed to consume all network bandwidth and be almost indistinguishable from legitimate traffic. Such was not the case in 1998, however. At the time, many denial of service attacks exploited weaknesses in network stacks or services, each form of which had a unique signature, hence we expect Snort to do well on these. Still, there were a limited number of flood style attacks, the most popular of which being the SYN-flood. While a Snort preprocessor could probably be built to detect such an attack, Snort does not come with one by default, and only includes rules to detect SYN-floods from clients (zombies) with a particular packet signature. As such, the bulk of the neptune (SYN-flood) attack packets that Snort does detect in the dataset are those targeted towards ports typically associated with services that may provide information leakage, such as SNMP. The degree of true versus false positive is related to the amount of legitimate usage of these services in the dataset.

Two denial of service attacks that Snort detects perfectly are the “back” attack against the Apache webserver, and the “land” attack against various TCP/IP stacks. Both are perfect examples of non-resource exhaustion DoS attacks that were more common in 1998 than they are today. Snort also detects almost 97% of ping of death (pod) packets with its “spp_frag2” preprocessor, with no false positives. This accounts for the bulk of the initial detection spike on DoS attacks in figure 2 (line with empty squares).

Snort did not fare as well on the smurf, syslog, and teardrop attacks. Only five of the 250133 smurf attacks were detected, and those were only detected because they generated an “ICMP port unreachable” response. Snort fires on “ICMP port unreachable” packets because such activity indicates probing activity; yet it also fires due to network outages, and accounts for a large number of the false positives observed. Generally, however, smurf is a resource exhaustion attack utilizing ICMP packets instead of SYN packets, so we expect Snort’s detection performance on it to be similar to the neptune attack described above, for the same reasons. The second, syslog, sends syslog messages from unresolvable IP addresses, which would be difficult to write a rule for (it would actually require a preprocessor with fairly high overhead). The failure to detect the teardrop attack is a little more difficult to explain. Supposedly, the teardrop attack, which misuses fragmentation in a UDP packet should be detected by Snort using the rule “1:270”, only that rule never fired in our tests with the DARPA dataset³.

³We couldn’t determine whether this was a problem with the dataset or Snort. The best way to determine that would be to launch the actual attack against a vulnerable machine (to verify that the attack is correct), and see if Snort detects it. Given that we didn’t have either the actual attack or a vulnerable system on hand, we did not pursue the matter further.

2.2.3 Probes

The probe activity in the dataset is all performed over the network, and most of that activity is noisy and undetectable by Snort, given the correct rules. In our experience, the default Snort rules have a very low threshold for detecting probe activity (for instance triggering on almost all ICMP activity), hence we expect it to do well in this category. The performance is shown by the line with the solid boxes in figure 2.

Overall, 16 rules detected over 20% of the probe connections. Eleven of the rules detected many of the probes from the portsweep, nmap, and satan scans with a negligible number of false positives. Two of the rules were specifically designed to detect probing activity, and the other nine were each designed to detect information leakage from a particular service. The next rule detected all ICMP ping activity, which allowed the majority of ipsweep connections to be detected (as well as a large amount of nmap activity), and also accounts for the sharp increase in false positives. Likewise, the following rule, designed to detect the attempted evasion of ID systems, finds some of the satan, portsweep, and ipsweep activity with a few thousand false positives. The remaining three rules all find a small number of the probe connections with a number of false positives due to legitimate use of the services they are designed to detect activity against.

While Snort's performance does not look particularly impressive when graphed, it succeeds in detecting part of every probe, which is sufficient for identifying a probe is taking place⁴. In fact, logging every connection associated with a probe will likely only serve to fill up a system's log files. What was troubling is that it only detected the ipsweep scans using a rule that fires on all ping activity, which generated a huge number of false positives on the DARPA data, and which we would expect to also happen in a real environment.

2.2.4 Remote To Local

The primary intent of Snort is to detect intrusions that happen over the network. As such, we would hope that it does well in the Remote To Local (R2L) category, especially given the time elapsed between the evaluation and the ruleset used, which should have allowed plenty of time to fine-tune the detection signatures. The results are shown by the line with empty circles in figure 2.

Indeed, we see that with just six rules we detect well over 10% of the malicious connections with a negligible number of false positives, and another five rules detect well over half the attacks; however, the false positive rate is unacceptably high. Notably, unlike the previous categories, the true detection versus false positive rate did not cumulatively exceed the 50% guess rate, although it came close.

Only one attack, phf, was perfectly detected with no false positives. Two of the attacks, dict and guest, were almost perfectly detected by rules designed to detect failed login attempts; yet these also produced a reasonable number of false positives, as we would expect them to in a real environment. We wouldn't

⁴It is not, however, sufficient if the IDS is used to identify packets to block.

expect Snort to do better on the dict attacks without a preprocessor, and while a rule could be written to detect the usage of guest accounts, this is probably better handled by a host-based detector. Both the spy and arez attacks were perfectly detected by rules designed to detect policy violations: the use of telnet and anonymous ftp servers, and given their legitimate use in the dataset, both rules produced a large number of false positives.

The attacks that had a low true positive rate were ftp-write, warezclient, and warezmaster. Some of these instances were detected by the aforementioned policy rules; however most of the connections that were detected, were detected by rules specifically written for these attacks, hence they had a low number of false positives. Generally, these attacks are difficult to write rules for as they don't violate a technical specification, but rather abuse legitimate services and use them in unintentional ways. The remaining R2L attack, multihop, was detected twice by a rule designed to detect many different types of attacks with only a single false positive, and once by the general rule for telnet access with the very high number of false positives, leaving six of its instances undetected.

2.2.5 User To Root

Generally, a User To Root (U2R) attack entails actions that happen solely on a local machine to elevate a user's privileges to that of the superuser. As such, the best way to detect them is via a host-based detector which may use inputs such as the syscall trace, changes to the filesystem, and process table. While such data was included as part of the evaluation, since Snort is a network-based IDS, it did not take advantage of these resources. Such attacks may be launched via a remote terminal session, as all apparently were for the evaluation (meaning it wasn't concerned with insider threats). If such a session is encrypted, then our primary hope of flagging it as intrusive comes from machine learning recognizing that the attributes of the connection (such as the source, time, packet interarrival times, etc.) is anomalous. If the connection is not encrypted, such as the Telnet sessions in the DARPA data, then we have some hope of detecting the attack based on a signature, such as the commands typed, or the output produced. Such signatures tend to be fragile, and we did not have high expectations for Snort's performance in this area.

As shown by the line with solid circles in figure 2, a small number of rules (five) were able to detect some effects of some of the U2R attacks, particularly rootkit, imap, ffb, and loadmodule with only a few false positives. Interestingly, the bulk of this initial spike is from a rule to detect BSD-style ICMP ping traffic, which detected 89% of the 254 rootkit attacks with only four false positives. This indicates that the particular rootkit used in the evaluation used its own ping packaged with it. Most of the rest of this initial spike was from rules designed explicitly to detect attack code and responses in terminal traffic, which interestingly produced five false positives. At the beginning of the slope into higher false positives, a couple of rules detected odd network traffic (failed logins and SYN-FIN packets) during a couple of the attacks, with a significant level of false positives. Most of the remaining attacks were detected with the policy

rule to detect all telnet traffic which has significantly more false positives on the test network than the number of attacks detected. Finally, a few remaining rootkit instances were detected with the afore mentioned (in the section on DoS attacks) rule that fires on all “Port Unreachable” packets. With a well tuned Snort ruleset (meaning one that has had rules that produce high rates of false positives removed) on a contemporary network (where most terminal sessions are encrypted), we would only expect to detect imap – which could also be classified as a remote to local attack – with any reliability.

2.3 Conclusion

While we initially thought that Snort would perform well on the DARPA IDS evaluation dataset, we found that it did not. Upon discovering this empirical result, we thought that it may be due to a failure in the dataset to model the attacks correctly, or perhaps because the attacks were so old that Snort no longer included the rules to detect them.

What we found instead was that the dataset only includes a very limited number of attacks that are detectable with a fixed signature. The majority of malicious connections in the DARPA dataset come from denial of service attacks and probing activity. While Snort has some capability to detect such attacks, such detections have not been the primary focus of its design. Interestingly, many of the advanced ID systems that were evaluated either as part of the DARPA evaluation, or that evaluated themselves using that data, fared very well on the DoS and probe attacks, whereas many had trouble with the low connection rate remote to local, and user to root attacks, where Snort performs best. As such, Snort should not be changed to detect some of these other attacks, but rather used in conjunction with another NID designed for such purposes. Snort had trouble detecting misuse of resources, such as the various warez attacks, where an anomaly based detector may fare better. While Snort performed impressively on the user to root attacks (although one must consider that many were detected solely because a telnet connection was used), we suspect that, given the widespread adoption of encrypted connections, that the days of NIDs detecting such attacks are numbered, and that a host-based intrusion detection approach is necessary.

Given that Snort detected most of the attacks that we would expect a signature-based detector to, and that many other attacks were detected through more erroneous means – policy rule violations, for example – we feel more comfortable saying that the DARPA IDS evaluation dataset properly models attacks, and can serve to help evaluate the true positive performance of a network intrusion detector for the types of attacks that it includes. We did not, however, get the same impression of its usefulness in evaluating the false positive performance. As McHugh observed, no evidence has been provided that the DARPA dataset accurately models a real network. In fact, casual observation shows that the data rate of the provided network data is far below that of even a trivial network used for business (as opposed to casual home users). This was actually an issue in our experiment, as apparently all, or at least a very signifi-

cant number, of the images transferred as part of their artificial HTTP activity were 0x0 sized images, which Snort kept alerting about, because these are typically used as “web-bugs” to track the surfing habits of users across the WWW. We discarded that rule as it didn’t provide any true positives; yet it illustrates how the artificial nature of the dataset prevents an objective evaluation of the false positives produced by an IDS. Conversely, we wonder how accurate the low-number of false positives with the BSD-style ping rule that detected most of the rootkit instances was. Generally speaking, we do not believe that the false positive rates reported herein have any bearing on Snort’s false positive performance in a real environment, which may be much worse, or much better.

In conclusion, we believe that any sufficiently advanced IDS should be able to achieve good true positive detection performance on the DARPA IDS evaluation dataset. Demonstrating such performance, however, is only necessary to show the capabilities of such a detector, *it is not sufficient*. In particular the intrusion detection community needs a more realistic dataset to evaluate the false positive rate of such systems. Further, there are many new threats that have emerged in the five years since the last evaluation (botnets, spyware, flash worms, etc), that a contemporary IDS needs to be able to detect, and which should be tested for in such a dataset. Additionally, a better dataset should include a realistic number of attacks which could be easily detected by a signature-based detector. How such a dataset can be produced and validated is an open research problem. We believe that even when these more advanced network ID systems are deployed, signature-based NIDs, such as Snort, will continue to play an important role in enterprise security architectures.

3 Previous work

There has been surprisingly little research on network simulation for the purposes of testing network based security systems. Most all of the network modeling and simulation work to date has been done by the network infrastructure community for the purposes of developing and testing more efficient queuing algorithms and new protocols. In this arena, the interest is primarily in flow rates and interarrival times. While these things are certainly important when simulating network traffic for security purposes – for instance to ensure that devices can handle normal network loads – it ignores much of the internals of the traffic, for example the application mix. When things like the application mix are considered, the focus is usually on the few applications that make up the bulk of network traffic flow, ignoring the low traffic applications that may be of considerable importance to security devices (such as RPC and SMB services).

When modeling the network flows for internetworking purposes, relatively statistical metrics have been sufficient to validate the models, and hence the correctness of the simulations created by those models. The advances in network modeling have been driven primarily by the use of better statistical measures of the network flows.

From the early days of computer networks, many assumptions were made

about network packet flows, primarily that traffic was dominated by bulk transfer operations, which moved a large quantity of data, that interactive applications produced fairly symmetric flows, and that packets followed a Poisson distribution model at any given point in the network. In 1992, Danzig et al. made a number of discoveries that advanced our understanding and ability to model networks. Specifically:

- Different applications produce different packet interarrival times.
- Interarrival characteristics need to be modeled at the application level since network infrastructure changes can change them.
- Interarrival times are only interesting for interactive applications since they will change for bulk transfer applications with network improvements.
- Bulk data is not necessarily large or unidirectional, and interactive traffic isn't symmetric in size in both directions.
- The distribution of network endpoint pairs differs between applications

Danzig et al. measured a large number of traffic characteristics in the course of this research including packet interarrival times and size, session bytes transferred (in each direction and bidirectionally), duration and packets transferred, items per connection for bulk and item sizes, network pair distribution, probability of concurrent connections to same network or host, connection rate by time of day, and MTU.

Despite the large of parameters that they considered, the work is primarily focused on the interests of the network engineering community. In particular, the authors only considered the data portion of the connection, and ignored packets that did not produce a significant load, such as establishment, tear down, ack-only packets, retransmitted packets, and anomalous packets. Such packets are important from a security perspective however, and are likely useful to the network protocol community as well.

The following year Leland et al. released the paper (expanded in 1994) that set the stage for network research for the next decade by showing that Ethernet packet traffic (including traffic introduced from the Internet) had self-similar – or fractal – interarrival times. Specifically, the statistical nature of the interarrival times followed the same long-tail distribution at multiple time scales.

While certainly an interesting result with unquestionable ramifications for router development, this unfortunately caused an almost singular focus on fractal patterns for interarrival times within the network modeling community:

1. Erramilli et al. (1994) showed how chaotic maps could be used to model and produce the self-similar distributions of packet interarrival times.
2. Paxson and Floyd (1995) established that Poisson processes were useful for modeling the interarrival times of user-initiated sessions, but that other session interarrival times, and the packet interarrival times within all WAN sessions, are best modeled using self-similar processes.

3. Park (1997) classifies the research into the causes of self-similarity. Evidence was found for multiple sources, including packet queuing by network stacks, protocol reliability techniques, application characteristics, and the nature of the data itself being transmitted. Park notes that source based simulation is necessary for proper generation of network traffic because direct packet generation fails to capture the interactions between components which creates the fractal pattern in the first place.
4. Sikdar and Vastola (2001) went on to provide a mathematical formalism for why TCP, in particular its retransmission features, can create inter-packet arrival times that follow a fractal pattern, even within an individual connection. In particular, they show that the self-similarity of a connection is directly proportional to the loss rate on the connection. They go on to hypothesize that this behavior may vary – however will not vanish – depending on the TCP implementation used.
5. Racz et al. (2003) expanded the work in (Sikdar and Vastola 2001) by looking at the dominate cause of self-similarity under different conditions. They found that synchronization between sessions was the dominate cause in low-congestion and low-loss networks. We assume that this is due to network queuing effects as noted in earlier papers. In the face of modest congestion, the TCP congestion avoidance protocol quickly becomes the dominate contributor to self-similarity. We find it interesting that Racz et al. looked only at the network layer and above, as the original self-similarity paper by Leland et al. considered all Ethernet traffic, hence implying that the self-similarity was an artifact of the data-link layer. Perhaps unknowingly, Leland et al. imply that the Ethernet congestion avoidance protocol is a cause of this property when they suggest that the congestion avoidance protocol for ISDN-B be carefully analyzed to be robust in the face of fractal patterned traffic. We wonder if congestion avoidance at the data link layer was only a significant contributor to self-similarity when Ethernet was used in a shared segment topology, as in Leland et al., as opposed to the switched topology most likely used by Racz et al.. An interesting test would be to repeat Racz et al.'s experiments on a data link protocol such as token ring. In any case, once the network starts having a high loss rate, TCP's resend mechanism takes over for the congestion avoidance protocol as the primary contributor to self-similarity.
6. In a departure from the network engineering community, physicists interested in the nature of scale-free networks have become interested in the Internet. In Yook et al. (2002) the authors show that topology of hosts on the Internet follows a fractal (self-similar) pattern⁵. While they don't investigate the result this has on the network traffic, one wonders if the self-similar nature of the network topology is one of the causes of self-similar interpacket times for network traffic.

⁵While the authors call it the “physical layout” of the Internet, link-layer protocols such as label-based switching abstract the network layer topology from the physical topology.

Floyd and Paxson (2001) followed up their seminal 1995 paper with an excellent examination of why simulating Internet traffic is so hard. The most valuable contribution of this paper is their list of invariants which have held for Internet traffic to date, and which are likely to continue to hold for the foreseeable future. Fortunately, it was here that we begin to see a break from a singular focus on self-similarity which, while certainly a factor in the list of invariants, is not the sole invariant. The paper also succinctly summarizes the important conclusions of the above research that preceded it.

Mellia et al. (2002) made a huge push to look at the network traffic as more than just the interarrival time between the sessions and packets, and the related packet size, session duration and bytes transferred, or loss rate. They put together a tool (Tstat) that can produce 80 different graphs or plots of network performance characteristics from a network trace. This was a significant advance to allow an analyst to see the effect of various TCP/IP settings on performance; however the focus was squarely on performance and not security, and it did not provide any means to automatically model the network such that new traces could be produced with the same characteristics. For example, no mention is made of measures of the use of IP options or overlapping IP fragments, either of which may be used to attempt to break certain TCP stack implementations. Further, there does not appear to be any correlation between various metrics and the source or source / dest pair, which would be useful to create a trace with the same characteristics, and identify hosts with conflicting measures, such as a host appearing to use both a 64 and a 128 TTL, which may indicate it's attempting to probe the network, or may just be acting as a network address translation gateway.

The most complete system for modeling and validating network traces is presented in (Lan and Heidemann 2002). The authors build application level source models by hand, then parameterize them with distributions automatically built from network traces. The traces produced when the models are used for simulation are then validated using quantitative and qualitative statistical measures. This work pulled together many positive qualities, such as:

- They recognized that the models were the same for different networks, and that it's the parameterization that dictates the unique characteristics.
- They demonstrated the importance of application level parameters (in this case, the distribution of object sizes in HTTP).
- They use multiple quantitative statistical comparisons (Wavelet Scaling Plot and Kolmogorov-Smirnov Goodness of Fit Test) to validate the generated traces.
- For the Kolmogorov-Smirnov test they used the most restrictive critical value of significance so as not to dictate a distribution.
- Instead of just measuring the packet interarrival time, they also used the packet size, session duration, size, and interarrival, and the per-host in-

terarrival and duration, as well as the protocol mix and traffic volume for comparison.

- Additional parameters were used for the models, specifically, TCP window size, Round Trip Time, and bottleneck bandwidth.

Nevertheless, their work left many questions unanswered:

- While the use of quantifiable statistical measures for comparison was a large leap ahead of the standard Hurst parameter estimation that most work used in the decade that preceded it, no justification was given for the use of the Kolmogorov-Smirnov test or the Wavelet Scaling Plot (although there is some precedent for the latter in the network engineering community).
- The presented results – particularly that the inbound and outbound traffic from a network, and the same network at two different times, and two different networks differ above the level of significance – are intuitive. Further, it seems convenient that the same statistical measures validate that the generated traces are below the threshold of significance for difference when compared to the real traces. What is missing is a demonstration of how different the traces must be before they are labeled as different. For instance, the authors show that a one hour trace starting at 14:00 is different from a one hour trace starting at 19:00. Is the one hour trace starting at 14:00 different from the one starting at 15:00? Intuitively, the same sort of activities should be happening on a network at those hours of the afternoon. Baring any reasonable explanation of why they should be different, if the statistical measures say that they're different, then we must question the usefulness of those techniques for trace comparison.
- They note that huge data sets do not exactly follow statistical distributions, so they use a random sample of 10000 data points for their comparisons (an accepted method in the statistical community). It seems that the outliers in network traffic are significant, and that ignoring them – especially if they are significant enough to skew a distribution – stands to lose vital characteristics of the network.
- In addition to the two quantitative comparisons, the authors did a qualitative comparison of the Cumulative Distribution Function (CDF); however humans have a tendency to find patterns where they don't exist. This did seem to serve well as a first order comparison to see if two distributions were obviously different.
- As the authors note, only HTTP and FTP were modeled, and a natural extension to their work would be to add additional protocols. We would be concerned, however, that modeling each protocol by hand does not scale with the growing number of application protocols in use on modern networks, and that failing to model infrequently used protocols, which

may be fine when generating traffic for network engineering purposes, will pose problems for the network security community, where an increase of an infrequently used protocol is typically indicative of a new attack. Indeed, the choice to model FTP, with its higher percentage of bytes transferred over DNS, with its higher percentage of packets and sessions, demonstrates this bias in the work as presented.

- Even within the HTTP and FTP models that they used, they chose not to model HTTP clients which use multiple connections per page or the FTP control channel, both of which are bound to have an adverse impact on the use of such models to generate traces for network security testing.
- We also find it odd that their generated HTTP/FTP traffic matched the real traffic when the real traffic was filtered down to just the HTTP and FTP connections. By their own reckoning, a source model is necessary as the individual session traces will behave differently in the presence or absence of other network traffic, so it seems that traces generated in the absence of non-HTTP/FTP traffic should differ from the real traces.

Most recently, Bartoletti and Tang (2005) demonstrated the ability to characterize services based on the differences between vectors to the centroids of clusters representing the services. These representative clusters are produced by clustering thousands of connections to that service into a handful of clusters. Each connection is represented by seven TCP session characteristics. The authors demonstrate that the differences between the centroids of the clusters fall below a standard deviation for clusters within the same service, and are significantly larger between clusters representing different services. Some services are obviously more similar (HTTP and HTTPS) than others (HTTP and FTP). The authors speculate that this method could be used to identify the type of traffic in a tunneled connection, however they do not demonstrate the use of the method for session identification, which is potentially the most interesting application of this approach. Unfortunately, the authors do not have a measure for how many connections are necessary to get a representative cluster. Put another way, they do not know how the method performs given a small number of connections. We would be interested to see how this method compares with our proposed methodology (below), particularly in the area of network (rather than per service) comparison.

4 Methodology for the Comparison of IP Network Traces

Given the lack of prior work to quantitatively compare two networks based on their network traces, we propose a new methodology to that end. This section will outline the goal of that methodology, how we will measure the success of the proposed methodology, the cases that we will use for deriving the relevant characteristics and weights used in the application of the methodology, the cases

that we will use for testing our success metrics, the actual data sources we will use for this research, and the methodology itself. We will conclude with the timeline for our proposed work, and the intended deliverable.

We intend for this methodology to be generally applicable for the comparison of any pair of computer network traces, however the characteristics and weights that are used for any given comparison will depend on the data sources, the types of networks being compared, size of the networks being compared, and the exact goal of the comparison. For this proof of concept, we compare the similarity of IP networks⁶ of 255 or less hosts based on tcpdump captures of their activity from the edge. The characteristics and weights we develop should be applicable to this end for contemporary networks. Over time these will change. For example, we expect that the shared segment Ethernet architectures of a decade ago would change some of the characteristics and weights from the switched architectures of today. Likewise, application of the methodology for a different goal would require different base datasets, for instance measuring the change of a network over time would likely base their characteristics and weights on data collected from inside the network, as opposed to the edge. Different data sources, for example router or firewall logs, will change the available characteristics available to the methodology, hence changing the weights, and potentially the usefulness of various characteristics. Obviously, different types of networks, for example ATM or packet-radio, will require different data sources and produce different characteristics and weights. We are less certain what impact network size will have on the characteristics and weights derived by this methodology, however we expect that significantly larger networks, such as those representing an entire research university or large ISPs will require different characteristics and weights to compare. This is an area for future research.

4.1 Goal

Be able to quantitatively measure the differences between two computer networks. Will produce a real number score, which we define to be between zero (no similarities) and one (perfectly similar), inclusive, for convenience. Will also produce a report of what the significant contributors to that score are (service mix, times, statistical distributions of characteristics, etc).

4.2 Success metrics

1. Network trace and anonymized version of same trace show no difference.
2. Same network at adjacent times (without major sociological difference) shows some minimal difference.
3. Same network at same time, different weeks, shows some minimal difference.

⁶When we mention IP networks, we mean IPv4 networks, as opposed to IPv6.

4. Same network at disparate times (on/off hours) shows more significant difference.
5. Similar networks show some difference at the same time, and different networks show larger difference (edu/edu vs edu/com).
6. Completely different traces (disjoint networks, disjoint times, disjoint services) show minimal similarity.

4.3 Base and test cases

We have 17 base cases, which are used to determine the proper set of characteristics and weights to use with the methodology. They are as follows:

1. One hour of data and anonymised version of that hour (should score 1.0)
2. One hour of data and the same hour repeated with the timestamps modified to appear to be one hour later (should score 1.0)
3. One hour of data and the same hour repeated one hour later with a few connections added, a few removed, most shifted slightly in time, some host pairs changed (should score approx 0.9)
4. The same pair with even more modifications (should score lower)
5. One hour of data from one network, and the same hour of data generated (by manually manipulating real data as necessary) to share no similarity (size, protocol mix, topology, etc) with the first network trace (score should approach 0.0 – it will not be zero unless one network has traffic and the other does not)
6. Two adjacent hours of traffic from the same network in mid-day (score should be > 0.75)
7. Two adjacent hours of traffic from the same network in off-hours (score should be > 0.75)
8. The same mid-day hour of traffic from the same network on adjacent weeks (score should be > 0.75)
9. The same off-hour of traffic from the same network on adjacent weeks (score should be > 0.75)
10. One mid-day hour and one off-hour of traffic from the same network (score should be lower than the two above scores)
11. One hour of data from one network, and the same hour of data from a different network that is similar in size, topology, policy, user base, etc, such that we should expect the traffic to be similar in composition (score should be > 0.5 and < 0.75)

12. One hour of data from one network, and the same hour of data from a different network that differs in a single significant way (size, topology, policy, user base, etc) from the first such that we should expect the traffic to share some similarities, but still be less similar in composition to the above (score should be > 0.25 and $<$ above)
13. One hour of data from one network, and the same hour of data from a significantly different network (score should be < 0.25)
14. One day of data from a network, and one day of data from the same network on the same day of the following week (score should be > 0.75)
15. One day of data from one network, and the same day of data from a similar network (score should be > 0.5 and < 0.75)
16. One day of data from one network, and the same day of data from a slightly different network (score should be > 0.25 and < 0.5)
17. One day of data from one network, and the same day of data from a very different network (score should be < 0.25)

While any number of trace pairs could be compared using the methodology once we determine the proper characteristics and weights, we use seven test cases, which correspond to the six success metrics:

1. One hour of data and anonymised version of that hour (should score 1.0)
2. Two adjacent hours of traffic from the same network in mid-day (score should be > 0.75)
3. One day of data from a network, and one day of data from the same network on the same day of the following week (score should be > 0.75)
4. One mid-day hour and one off-hour of traffic from the same network (score should be > 0.5 and < 0.75)
5. One day of data from one network, and the same day of data from a similar network (score should be > 0.5 and < 0.75)
6. One day of data from one network, and the same day of data from a slightly different network (score should be > 0.25 and < 0.5)
7. One day of data from one network, and the same day of data from a very different network (score should be < 0.25)

4.4 Trace pairs

We will use a pair or pairs of network traces to first apply our methodology on each of the base cases to derive the relevant characteristics and weights, then we will use different pairs of network traces to apply our methodology to test how well it achieves the above goal.

We will only need a single pair for the first five base cases, as the results of those comparisons should not vary between networks (they are, after all, artificial). For the remaining 12 base cases, we should use three trace pairs each to eliminate the possibility of skew due to outliers in any given pair. Specifically, if any given pair is more or less similar than we think they should be, perhaps because we traced one organization while doing monthly network vulnerability scans, then the other two trace pairs for that set will allow us to detect that deviation and correct for it.

We will then use a single trace pair for each of the seven test cases. These pairs will most likely come from the same networks as the pairs for the base cases, on adjacent days. For example, we might collect traces on the 8th and 15th of the month from a network for deriving appropriate characteristics and weights, then test for success by using traces on the 9th and 16th on the same network.

Examples of pairs of similar networks which might be used:

1. Two residential DSL networks
2. Two Linodes (Virtual Linux Colocated Servers)
3. The CS network at Davis and the CS network at another university such as Purdue or Georgia Tech
4. Two commercial DSL networks

Examples of pairs of slightly different networks which might be used include:

1. A residential DSL network and a residential cable-modem network
2. The 222 and 223 networks at LLNL which differ primarily in how densely populated they are
3. The Davis CS network and another Davis departmental network, such as EE

Examples of pairs of significantly different networks which might be used:

1. A residential DSL network and a commercial frame-relay network
2. The 223 network at LLNL and a commercial frame-relay network
3. The 223 network at LLNL and a Linode
4. The Davis CS network and a commercial frame-relay network

The author has a residential DSL network and a Linode⁷, which will allow for analysis of data from the same network to begin immediately, while we work on obtaining network traces from other sites.

⁷Linode is a company which runs numerous virtual Linux servers on a smaller set of real servers in collocation facilities.

We have begun looking into the use of LLNL networks for this research. There is past precedent for this type of data collection for research purposes, given that the data is strictly protected, that only summaries of the data are used for publication, and review and release is performed on such publications. Final approval for data collection would need to be made by LLNL's cybersecurity officer and the Laboratory's legal consul.

We are less sure of what the availability of data from Davis departmental networks, such as the CS or EE networks would be, however we expect that similar protections and approvals would need to be made. Further guidance in this area would be appreciated. Likewise, we don't know what the availability of data from a school similar to Davis (such as Georgia Tech or Purdue) would be, however we intend to leverage our contacts at those schools to find out and ultimately, hopefully obtain data.

We do not have any specific companies in mind for collection of network traces from commercial networks; however, we believe that we should be able to find a company that is willing to cooperate with this research, given the same sort of data protection and legal agreements discussed above. We would be inclined to solicit the cooperation of companies where Davis Security Lab alumni work (as most of them would have an interest in the end-product of this work), however we fear that such companies tend to be research and development oriented, and hence might appear too similar to one of the LLNL or academic networks.

We are interested in the activity of the network (ingress and egress) at the edge, and as such will collect traces from all interfaces between a network and the rest of the world (either the Internet directly, or a larger network, such as the interface between a single department and the rest of a campus), and collate the traces together based on timestamp. This will require clock synchronization, preferably through the Network Time Protocol (NTP).

If a network uses a firewall, we would like to get simultaneous traces from both sides of the firewall. All of the above data sources will be a class-C or smaller since we expect that even at that size, one day of packet headers will be quite large. We would like to collect the same one day (midnight PST to midnight PST) of data from all of the above networks. Some of the data sources (the Linodes in particular) will just collect data by running tcpdump in promiscuous mode directly. The other traces will be collected with bare-bone Linux machines with GigE interfaces running tcpdump in promiscuous mode with the interfaces up, but not configured (so they will not contribute traffic to the network). All the machines will be clock synced and have clock drift adjusted for by running NTP with them networked together before being partitioned out to the collection sites. These machines will either be plugged into a spanning port from the edge router, or spanning ports on the switches on either side of the network firewall. In the event that the switch(s) do not have spanning ports, we will introduce a hub for 10/100 networks, and for GigE networks we will either put two NICs in the Linux box to record and pass-through all traffic (in which case, it will act like a two-port layer-2 switch), or introduce a simple switch with a spanning port between the device and the switch it normally plugs into, and

then plug into that spanning port.

4.5 Methodology

4.5.1 Derivation of Characteristics and Weights

1 Define first-order characteristics The first step in the methodology is to define the possible first order characteristics that we can derive from the data sources. For our application, we have compiled the following list, based on the list of IP network characteristics in (Brugger 2005):

1. The classic interarrival time based on number of packets and connections (continuous)
2. Distributions based on number of packets, connections, and bytes transferred:
 - (a) Seconds after the minute (from the timestamp) (continuous)
 - (b) Minutes after the hour (from the timestamp) (continuous)
 - (c) Hours after midnight UTC (from the timestamp) (continuous, essential)
 - (d) Hours after midnight Local (from the timestamp) (continuous, essential)
 - (e) Day of the week (from the timestamp) (continuous, essential)
 - (f) Source IP (discrete, also try as continuous with the data sorted by rate, essential)⁸
 - (g) Destination IP (discrete, also try as continuous with the data sorted by rate, essential)
 - (h) Protocol/dest port (or type/code for ICMP) (discrete, essential)

For TCP/UDP connections:

- (i) Source port (discrete, also try as continuous)
3. Distributions based on the number of TCP connections:
 - (a) Duration (continuous)
 - (b) Frag packet rate (continuous)
 - (c) Wrong frag packet rate (continuous)
 - (d) Urgent packet rate (continuous)
 - (e) Resend packet rate (continuous)
 - (f) Wrong resend packet rate (continuous)
 - (g) Duplicate ACK packet rate (continuous)
 - (h) Hole rate (continuous)

⁸While the client IPs will vary between networks, certain IPs – such as those of popular Internet servers and prolific attackers – will be invariant, hence the rationale to model IPs discretely; however, the large number of variant hosts lends to the intuition that this characteristic may be better to model continuously; both ways can be attempted and neither, either, or both may be used.

- (i) Wrong data packet size rate (continuous)
 - (j) Percent data packets (continuous)
 - (k) Percent control packets (continuous)
 - (l) Window size (continuous)
4. Distribution of the number of packets versus the number of packets over the past w seconds for each packet (continuous)
 5. Distribution of percentage of packets to privileged services out of the past n packets for each packet (continuous)
 6. Distribution of the number of packets to privileged services versus the number of packets over the past w seconds for each packet (continuous)
 7. Distribution of percentage of packets to unprivileged services out of the past n packets for each packet (continuous)
 8. Distribution of the number of packets to unprivileged services versus the number of packets over the past w seconds for each packet (continuous)
 9. Distribution of the number of connections versus the number of connections over the past w seconds for each connection (continuous)
 10. Distribution of the number of connections versus the following, calculated over the past n connections or w seconds, for each connection:
 - (a) Number of packets (continuous)
 - (b) Number of SYN flags (continuous)
 - (c) Number of establishment errors (continuous)
 - (d) Number of other errors (continuous)
 - (e) Number of RST flags (continuous)
 - (f) Number of FIN flags (continuous)
 - (g) Number of PSH flags (continuous)
 - (h) Number of connections to privileged services (continuous)
 - (i) Number of packets to privileged services (continuous)
 - (j) Number of connections to unprivileged services (continuous)
 - (k) Number of packets to unprivileged services (continuous)
 - (l) Average duration (continuous)
 11. Distribution of number of packets send and received, and bytes sent and received per connection (continuous)
 12. Distribution of packet sizes (continuous)
 13. Distribution of TTL (continuous, also try as discrete)
 14. The total number of incoming and outgoing packets, connections, and bytes.

We will investigate both using and not using virtual UDP connections, where the UDP packets between the same hosts for the same service with a maximum interpacket latency below some timeout threshold (such as 150 seconds) is treated as a single logical connection.

Now we run each of our base case test pairs through the following steps:

2 Determine distribution function for each characteristic For each of the above characteristics, build its distribution for the first trace:

- The total number characteristics will just be kept as a single value.
- For the characteristics that are labeled as discrete, we will just keep a table of values.
- For the continuous characteristics, we will begin with a least-squares quadratic function fit to the data points. The prior work shows that we will need to use more complex representations (such as a fractal distribution, Fourier or Principle Component analysis, wavelet modeling, etc) for some of the characteristics, so we will use such distributions as our analysis and expertise dictates.

3 Build the distributions for the second trace Build the distribution for the same characteristics of the second trace. If two networks differ only by size, this will be reflected in the differences between the total packet and byte counts, so we don't want it to also be reflected in the differences between distributions, such as the distribution of packets to privileged services over the past w seconds. To correct for this, we will use $\frac{\tau_1}{\tau_2}$ as a multiplier for any distributions (or the individual measurements used to build a distribution) built for the second trace⁹. This will allow us to compare distributions without worrying about one distribution having twice as much traffic as the other distribution. A potential problem with this approach that we will need to remain vigilant of, is that it will also scale traffic that is invariant with load (for instance an automated process that makes a single connection every hour). An alternative approach would be to scale based on the number of active hosts on the target network.

4 Determine similarity function for each characteristic Build the distribution for the same characteristic of the second trace. Now we can calculate the similarity between the two distributions for that characteristic:

- For the total counts, we will use the formula

$$1 - \frac{|x_1 - x_2|}{x_1 + x_2}$$

as the normalized similarity between the two. This formulation is like the percentage similarity (1 - the percentage difference) between

⁹where τ_1 and τ_2 are total packet counts (in and out) for distributions based on packet counts, bytes for distributions built on bytes, etc

the measurements, with the variation that we divide by the sum of the values, rather than the mean of the values, as this will naturally constrain our values in the range $[0, 1]$ rather than $[0, 2]$ (hence, “normalized” similarity), as long as all of our measurements are positive numbers (which in the domain of network measurements, they should be).

- For discrete characteristics, we will use the average of our normalized similarity between the measurements:

$$1 - \frac{\frac{|x_{(1,1)} - x_{(1,2)}|}{x_{(1,1)} + x_{(1,2)}} + \frac{|x_{(2,1)} - x_{(2,2)}|}{x_{(2,1)} + x_{(2,2)}} + \dots + \frac{|x_{(n,1)} - x_{(n,2)}|}{x_{(n,1)} + x_{(n,2)}}}{n}$$

where $|x_{(i,1)} - x_{(i,2)}|$ is the absolute value of the difference between the i^{th} measurement of the first and second traces.

- For continuous characteristics, the similarity calculation will depend on the function being used to model the data. As long as we’re using the quadratic function discussed above, we should be able to sample the curve at given points, and then use the same similarity formula as we do with the discrete characteristics to find the normalized similarity between the two curves. More complex representations will require further analysis to determine the proper similarity method.

Once we’ve performed the above steps for each of the trace pairs for the base cases, we must consider the usefulness of each characteristic, given its behavior across all the base cases.

5 Find the weights for each characteristic The normalized similarity for each characteristic should perform as noted for all trace pairs in all base cases. This may require that the normalized similarity is multiplied by an adjustment factor. Such an adjustment factor may be non-linear. The determination of this adjustment factor will need to be made by an analyst, taking into consideration the target similarity value for each test case. The normalized similarity will be known as the scaled similarity once multiplied by the adjustment factor.

6 Drop non-differentiating characteristics If we are unable to find an adjustment factor which allows the normalized similarity to reach the target similarity value, for a significant number of base cases, we must determine if that is

1. due to a failure of the distribution algorithm used (easiest to tell if it tells us that two traces are similar when they are not; however, equally likely in other situations),
2. a failure for a characteristic to change between different traces (be non-differentiating), or
3. or a data failure in two traces being more or less similar in some respect than intended.

If a characteristic does not differentiate a significant number of base cases (either because it is invariant, or it is random), then drop it.

- 7 Verify performance using all characteristics** Take the scaled similarity for all the retained characteristics and average (find the arithmetic mean of) them. The average scores should be consistent with the intended scores. If it is not, then steps 5 and 6 (and possibly 2 through 4) will need to be revisited.

Now we will repeat the above steps for higher order characteristics.

- 8 Determine second-order characteristics and weights** For any of the first order characteristics that were useful, repeat steps 2 through 7 above, treating it as a second order characteristic to each member of the set of essential characteristics (specified in the above list of characteristics). For example, if the distribution of TCP connection wrong resend rates is useful, then try it as a second order characteristic, such as the distribution of TCP connection wrong resend rates on a per service (dest port) or source IP basis. This will allow us to compare if wrong resend rates on a network are due to a particular service or host. If the second order characteristic proves useful, integrate it with the others as above. We choose to consider only higher order characteristics of lower order ones which proved useful, and limited them to be higher order to the essential characteristics only as a way of limiting our search space in this combinatorial problem. Granted, there is the possibility that there might be a higher-order characteristic this doesn't find, for instance between the relationship between the number of packets in the past n connections and the distribution of duration times in a given trace; however, we are more likely to find coincidences that have no basis in reality, or obvious relationships that provide no additional insight.

- 9 Determine higher-order characteristics and weights** We will continue with this process for higher order characteristics as long as useful characteristics are found. Expanding the above example, if the distribution of TCP connection wrong resend rates on a per service basis is useful, we will consider the TCP connection wrong resend rate per service per hour after midnight UTC or per destination IP.

4.5.2 Comparison of IP Network Traces

We are finally ready to use the characteristics and weights derived in the above steps to actually compare two network traces.

- 1 Build distributions of characteristics** Use the function determined in step 2, above, to build the distributions for all characteristics in each trace. Adjust the distributions of the characteristics in the second trace by any multiplier determined in step 3, above.

- 2 Find the scaled similarities for each characteristic** Use the similarity function for each characteristic, determined in step 4, above, to find the normalized similarity between each characteristic. Apply the scaling factor determined in step 5, above, to each normalized similarity.
- 3 Generate overall similarity measure** Average (find the arithmetic mean) the scaled similarities to determine the overall similarity measure between the two traces.
- 4 Generate report** The order of the scaled similarities between characteristics, from lowest to highest, determines which were the most significant contributors for the differences between the two traces. These contributors, plus the overall similarity measure, will be compiled into a user-readable report.

4.6 Timeline

The following timeline is based on the premise that this research will be approved by 1 January 2006 – any delay in approval beyond that date will likely push the following dates back by a time commensurate with the delay.

- 7 January 2006** Modification of data to generate trace pair for base case 2
- 15 January 2006** Modification of data to generate trace pair for base case 3
- 23 January 2006** Modification of data to generate trace pair for base case 4
- 31 January 2006** Modification of data to generate trace pair for base case 5
- 31 January 2006** Selection of data collection sites
- 7 February 2006** Build all six counts (steps 2 and 3) for the five trace pairs of the first five base cases
- 14 February 2006** Find the normalized similarities (step 4) for the six count characteristics for the five trace pairs of the first five base cases
- 21 February 2006** Find the scaled similarities (step 5) for the six count characteristics for the five trace pairs of the first five base cases; however, any weights we find here are tentative until we have all the trace pairs for all the base cases available (this will essentially provide practice in determining the weights across multiple base cases)
- 28 February 2006** Build all 11 discrete distributions (steps 2 and 3) for the five trace pairs of the first five base cases
- 7 March 2006** Find the normalized similarities (step 4) for the 11 discrete distributions for the five trace pairs of the first five base cases

- 23 March 2006** Find the scaled similarities (step 5) for the 11 discrete distributions for the five trace pairs of the first five base cases; however, any weights we find here are tentative until we have all the trace pairs for all the base cases available (this will essentially provide practice in determining the weights across multiple base cases)
- 31 March 2006** Memorandum of Understanding, and other legal agreements with data collection sites to collect their data and publish summaries of their anonymised data
- 30 April 2006** Data collected from sites
- 31 May 2006** Build all 72 continuous distributions (steps 2 and 3) for the five trace pairs of the first five base cases – this step will be time consuming since it will require analysis to determine what the proper function is to model each characteristic
- 31 July 2006** Find the normalized similarities (step 4) for the 72 continuous distributions for the five trace pairs of the first five base cases – this step will also be time consuming since it will require analysis to determine what the proper similarity function is for each continuous distribution model
- 15 August 2006** Find the scaled similarities (step 5) for the 72 continuous distributions for the five trace pairs of the first five test cases; however, any weights we find here are tentative until we have all the trace pairs for all the base cases available (this will essentially provide practice in determining the weights across multiple base cases)
- 23 August 2006** Build all six counts (steps 2 and 3) for the 36 trace pairs of the last 12 base cases
- 31 August 2006** Find the normalized similarities (step 4) for the six count characteristics for the 36 trace pairs of the last 12 base cases
- 15 September 2006** Find the scaled similarities (step 5) for the six count characteristics for all 41 trace pairs for all the base cases
- 30 September 2006** Drop any of the count characteristics which are non-differentiating (step 6)
- 15 October 2006** Build all 11 discrete distributions (steps 2 and 3) for the 36 trace pairs of the last 12 base cases
- 31 October 2006** Find the normalized similarities (step 4) for the 11 discrete distributions for the 36 trace pairs of the last 12 base cases
- 22 November 2006** Find the scaled similarities (step 5) for the 11 discrete distributions for all 41 trace pairs for all the base cases
- 7 December 2006** Drop any of the discrete distribution characteristics which are non-differentiating (step 6)

- 23 January 2007** Build all 72 continuous distributions (steps 2 and 3) for the 36 trace pairs of the last 12 base cases
- 21 February 2007** Find the normalized similarities (step 4) for the 72 continuous distributions for the 36 trace pairs of the last 12 base cases
- 23 March 2007** Find the scaled similarities (step 5) for the 72 continuous distributions for all 41 trace pairs for all the base cases
- 30 April 2007** Drop any of the continuous distribution characteristics which are non-differentiating (step 6)
- 31 May 2007** Verify that the average similarity for all 41 trace pairs for all the base cases falls within the target score for each base case, or adjust as necessary (step 7)
- 15 June 2007** Determine second-order characteristics (step 8-1)
- 15 July 2007** Build distributions of second-order characteristics for all 41 trace pairs of the 17 base cases (steps 8-2 and 8-3)
- 15 August 2007** Find the normalized similarities (step 8-4) for the second-order characteristics for all 41 trace pairs of the 17 base cases
- 15 September 2007** Find the scaled similarities (step 8-5) for the second-order characteristics for all 41 trace pairs of the 17 base cases
- 30 September 2007** Drop and of the second-order characteristics which are non-differentiating (step 8-6)
- 30 December 2007** Repeat steps 1 through 7 for as many higher-order characteristics as we can process in a three-month period (step 9)
- 31 January 2008** Build distributions for all characteristics for all seven of the trace pairs corresponding the the seven test cases (comparison step 1)
- 28 February 2008** Find the scaled similarities for all the characteristics for all seven of the trace pairs corresponding the the seven test cases (comparison step 2)
- 7 March 2008** Find the average similarity score for all seven of the trace pairs corresponding the the seven test cases (comparison step 3)
- 31 March 2008** Generate a report of the average similarity score and significant differences between each trace in the seven trace pairs corresponding to the seven test cases (comparison step 4)
- 30 April 2008** Compilation of results into dissertation form (much of this work will be done along the way, this is mostly clean-up and covering anything that was missed)

31 May 2008 Analysis of how well the individual success metrics and overall goal were achieved by the proposed methodology, and any additional verbiage to explain results

30 June 2008 Formatting and editing of final deliverable (dissertation)

4.7 Deliverable

The final deliverable of this research will be a dissertation which will analyze how well the above methodology fulfilled the success metrics, and hence the overall goal, laid out at the start of this section. To do so, it will provide detail on:

1. The characteristics which proved useful and the degree to which each agreed with the defined difference between each pair of datasets (as determined by the weights determined on each characteristic).
2. The characteristics that did not prove useful, and an analysis of why they did not work.
3. How the characteristics – particularly those represented by a continuous distribution – were modeled.
4. What functions were used to generate the similarity measure for each characteristic.
5. Examples of the final reports generated for the test cases.
6. An analysis of how well the methodology works and its applicability for future work.

Along the way, we expect that some of the intermediate results, particularly the applicability of various characteristics to differentiate network traces, and the degree to which the similarities and differences between networks can be quantified using network traces, to provide useful material for peer reviewed publications, however such publications will be viewed as an added bonus to the core deliverable.

5 Conclusion

Our research to date has shown that the DARPA IDS evaluation dataset is insufficient for intrusion detection research. While many deficiencies have been cited, the primary problem that we found was that the background traffic was not validated to be a proper emulation of real network traffic, which is necessary to evaluate the false positive rates of IDSs. Such a validation was not possible because there are no published methodologies to quantitatively compare network traffic at the header level with sufficient depth for network security research. The creation of such a methodology is necessary to advance the state

of the art of network intrusion detection research, and will be a boon to other areas of network intelligence, including the comparison of attacker MOs, network evolution research and management, protocol impact analysis, and passive network classification and comparison.

We propose the creation of a methodology which will provide a quantitative comparison between two network traces, based on the packet header and connection characteristics. We will implement such a methodology and demonstrate its correctness through its application to numerous datasets as outlined in this paper.

6 Disclaimer

What do you get when you cross a lawyer and a bureaucrat? The following: Some of the material in this document was prepared as an account of work sponsored by an agency of the United States Government and released as UCRL-CONF-214731. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

7 Acknowledgments

Some of the work in this paper was performed under the auspices of the United States Department of Energy by Lawrence Livermore National Lab under contract number W-7405-ENG-48. The author wishes to thank Jed Chow for his assistance with the evaluation of Snort on the DARPA dataset, the anonymous conference reviewers and Marcey Kelley for their feedback on the Snort evaluation, and Dr. Matt Bishop for his feedback on this paper.

References

- Bartoletti, A. and N. A. Tang (2005, 1 April). Characterizing network services through cluster-set variations. Technical Report UCRL-TR-211020, University of California, Lawrence Livermore National Laboratory, Livermore, CA.

- Brugger, S. T. (2005). Data mining methods for network intrusion detection. *ACM Computing Surveys*. In review for publication.
- Caswell, B. and M. Roesch (2004, 16 May). Snort: The open source network intrusion detection system. <http://www.snort.org/>.
- Danzig, P. B., S. Jamin, R. Cáceres, D. J. Mitzel, and D. Estrin (1992, March). An empirical workload model for driving wide-area TCP/IP network simulations. *Journal of Internetworking* 3(1), 1–26.
- Erramilli, A., R. P. Singh, and P. Pruthi (1994, 6–10 June). Chaotic maps as models of packet traffic. In *Proc. 14th Int. Teletraffic Cong.*, Volume 1, North-Holland, pp. 329–338. Elsevier Science B.V.
- Floyd, S. and V. Paxson (2001, August). Difficulties in simulating the Internet. *IEEE/ACM Trans. Networking* 9, 392–403.
- Lan, K.-C. and J. Heidemann (2002, July). Rapid model parameterization from traffic measurements. *ACM Trans. on Modeling and Computer Simulation* 12(3), 201–229.
- Ledesma, S. and D. Liu (2000, 21–25 August). A fast method for generating self-similar network traffic. In *Int. Conf. on Communication Technology (WCC-ICCT) Proc.*, Volume 1, Beijing, pp. 54–61.
- Leland, W. E., M. S. Taqqu, W. Willinger, and D. V. Wilson (1994, February). On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Networking* 2, 1–15.
- Lippmann, R. P. and R. K. Cunningham (1999, 7–9 September). Improving intrusion detection performance using keyword selection and neural networks. In *Proc. of the Conf. on Recent Advances in Intrusion Detection (RAID) '99*, West Lafayette, IN.
- Lippmann, R. P., D. J. Fried, I. Graf, J. W. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. K. Cunningham, and M. Zissman (2000, January). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proc. of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA. IEEE Computer Society Press.
- Lippmann, R. P., J. W. Haines, D. J. Fried, J. Korba, and K. J. Das (2000, October). The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks* 34, 579–595.
- Mahoney, M. V. and P. K. Chan (2003, 8–10 September). An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In G. Vigna, E. Jonsson, and C. Krügel (Eds.), *Proc. 6th Intl. Symp. on Recent Advances in Intrusion Detection (RAID 2003)*, Volume 2820 of *Lecture Notes in Computer Science*, Pittsburgh, PA, pp. 220–237. Springer.
- McHugh, J. (2000). Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by

- Lincoln Laboratory. *ACM Trans. Information System Security* 3(4), 262–294.
- Mellia, M., A. Carpani, and R. L. Cigno (2002, November). Measuring IP and TCP behavior on edge nodes. In *Proc. IEEE Globecom 2002*, Taipei. IEEE.
- Park, K. (1997, 7–10 December). On the effect and control of self-similar network traffic: A simulation perspective. In *Proc. of the 1997 Winter Simulation Conference*, pp. 989–996.
- Paxson, V. and S. Floyd (1995, June). Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. Networking* 3(3), 226–244.
- Pickering, K. J. (2002, March). Evaluating the viability of intrusion detection system benchmarking. University of Virginia Undergraduate Thesis.
- Racz, P. I., T. Matsuda, and M. Yamamoto (2003, 28–30 August). Contribution of the application, transport and network layers to the self-similarity of Internet traffic. In *Proc. of 2003 IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*, Volume 2, pp. 760–763. IEEE.
- RTI International (2005). PREDICT: Protected REpository for the Defense of Infrastructure against Cyber Threats. <http://www.predict.org/>.
- Sikdar, B. and K. S. Vastola (2001, 21–23 March). The effect of TCP on the self-similarity of network traffic. In *Proc. of the 35th Conf. on Information Sciences and Systems*, Baltimore, MD.
- Willinger, W., V. Paxson, and M. S. Taqqu (1998). *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, Chapter Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. Boston: Birkhäuser.
- Yook, S.-H., H. Jeong, and A.-L. Barabási (2002, 15 October). Modeling the Internet’s large-scale topology. *Proc. of the Nat’l Academy of Sciences* 99(21), 13382–13386.

Appendix: Snort rule performance

The following five tables show the cumulative true and false positive counts for the Snort rules that detected at least one attack that no other rule did. The rules are ordered based on the true positive to false positive ratio for each rule, eliminating true positives for attacks which were already matched. The five tables present the rule performance for all attacks, and for each of the four types of attacks.

Table 1: Cumulative results by Snort rule. Each attack is detected at most once. Rules are sorted in descending order of TP:FP ratio.

Snort Rule	True Positives	False Positives
[113:1:1] (spp_frag2) Oversized fragment, probable DoS	10138	0
[1:1420:11] SNMP trap tcp	11643	0
[1:1418:11] SNMP request tcp	13147	0
[1:1421:11] SNMP AgentX/tcp request	14648	0
[1:1445:5] POLICY FTP file_id.diz access possible warez site	14792	0
[1:613:6] SCAN myscan	14852	0
[1:504:7] MISC source port 53 to <1024	14872	0
[1:598:12] RPC portmap listing TCP 111	14878	0
[1:332:8] FINGER 0 query	14883	0
[1:323:5] FINGER root query	14888	0
[1:1147:7] WEB-MISC cat access	14893	0
[1:330:9] FINGER redirection attempt	14898	0
[1:584:11] RPC portmap rusers request UDP	14902	0
[1:503:7] MISC Source Port 20 to <1024	14906	0
[1:612:6] RPC rusers query UDP	14910	0
[1:335:5] FTP .rhosts	14912	0
[1:359:5] FTP satan scan	14913	0
[1:489:7] INFO FTP no password	14914	0
[1:547:6] POLICY FTP 'MKD ' possible warez site	14915	0
[1:621:7] SCAN FIN	16223	1
[1:546:6] POLICY FTP 'CWD ' possible warez site	16512	2
[1:1156:9] WEB-MISC apache directory disclosure attempt	18793	40
[1:368:6] ICMP PING BSDtype	19019	44
[1:1251:6] INFO TELNET Bad Login	19937	132
[1:1882:10] ATTACK-RESPONSES id check returned userid	19946	133
[1:527:8] BAD-TRAFFIC same SRC/DST	19981	140
[111:2:1] (spp_stream4) possible EVASIVE RST detection	25526	2142
[1:384:5] ICMP PING	36030	7780
[1:652:9] SHELLCODE Linux shellcode	36032	7782
[1:498:6] ATTACK-RESPONSES id check returned root	36034	7784
[1:1417:9] SNMP request udp	36035	7787
[1:1419:9] SNMP trap udp	36036	7790
[1:716:12] TELNET access	36097	10829
[1:624:7] SCAN SYN FIN	36099	10962
[1:402:7] ICMP Destination Unreachable Port Unreachable	36107	12089
[1:553:7] POLICY FTP anonymous login attempt	36108	14057

Table 2: Cumulative results on denial of service attacks by Snort rule. Each attack is detected at most once. Rules are sorted in descending order of TP:FP ratio.

Snort Rule	True Positives	False Positives
[113:1:1] (spp_frag2) Oversized fragment, probable DoS	10133	0
[1:1420:11] SNMP trap tcp	11624	0
[1:1418:11] SNMP request tcp	13114	0
[1:1421:11] SNMP AgentX/tcp request	14604	0
[1:613:6] SCAN myscan	14664	0
[1:504:7] MISC source port 53 to <1024	14684	0
[1:503:7] MISC Source Port 20 to <1024	14688	0
[1:1156:9] WEB-MISC apache directory disclosure attempt	16969	38
[1:527:8] BAD-TRAFFIC same SRC/DST	17004	45
[111:2:1] (spp_stream4) possible EVASIVE RST detection	21604	2047
[1:402:7] ICMP Destination Unreachable Port Unreachable	21609	3174

Table 3: Cumulative results on probes by Snort rule. Each attack is detected at most once. Rules are sorted in descending order of TP:FP ratio.

Snort Rule	True Positives	False Positives
[1:1418:11] SNMP request tcp	14	0
[1:1420:11] SNMP trap tcp	28	0
[1:1421:11] SNMP AgentX/tcp request	39	0
[1:598:12] RPC portmap listing TCP 111	45	0
[1:323:5] FINGER root query	50	0
[1:330:9] FINGER redirection attempt	55	0
[1:332:8] FINGER 0 query	60	0
[1:584:11] RPC portmap rusers request UDP	64	0
[1:612:6] RPC rusers query UDP	68	0
[1:359:5] FTP satan scan	69	0
[1:621:7] SCAN FIN	1377	1
[1:384:5] ICMP PING	11881	5639
[111:2:1] (spp_stream4) possible EVASIVE RST detection	12819	7641
[1:1417:9] SNMP request udp	12820	7644
[1:1419:9] SNMP trap udp	12821	7647
[1:716:12] TELNET access	12832	10686

Table 4: Cumulative results on remote to local attacks by Snort rule. Each attack is detected at most once. Rules are sorted in descending order of TP:FP ratio.

Snort Rule	True Positives	False Positives
[1:1445:5] POLICY FTP file_id.diz access possible warez site	144	0
[1:1122:5] WEB-MISC /etc/passwd	149	0
[113:1:1] (spp_frag2) Oversized fragment, probable DoS	154	0
[1:335:5] FTP .rhosts	156	0
[1:547:6] POLICY FTP 'MKD ' possible warez site	157	0
[1:546:6] POLICY FTP 'CWD ' possible warez site	446	1
[1:1251:6] INFO TELNET Bad Login	1362	89
[1:1882:10] ATTACK-RESPONSES id check returned userid	1366	90
[1:716:12] TELNET access	1385	3129
[1:553:7] POLICY FTP anonymous login attempt	1386	5097
[111:2:1] (spp_stream4) possible EVASIVE RST detection	1387	7099

Table 5: Cumulative results on user to root attacks by Snort rule. Each attack is detected at most once. Rules are sorted in descending order of TP:FP ratio.

Snort Rule	True Positives	False Positives
[1:489:7] INFO FTP no password	1	0
[1:368:6] ICMP PING BSDtype	227	4
[1:1882:10] ATTACK-RESPONSES id check returned userid	232	5
[1:652:9] SHELLCODE Linux shellcode	234	7
[1:498:6] ATTACK-RESPONSES id check returned root	236	9
[1:1251:6] INFO TELNET Bad Login	238	97
[1:624:7] SCAN SYN FIN	240	230
[1:716:12] TELNET access	268	3269
[1:402:7] ICMP Destination Unreachable Port Unreachable	271	4396

Appendix: Attack detection performance

The following table summarizes the true positive and false negative performance for all of the connections that were part of an attack. The true positives are the number of attack connections that Snort alerted on with at least one of the above rules. The false negatives are the number of attack connections that Snort failed to alert on.

Table 6: True positive and false negative performance per attack type

Attack Type	True Positives	False Negatives
anomaly	9	0
back	2281	0
dict	881	0
dict_simple	1	0
eject	11	0
eject-fail	1	0
ffb	5	5
ffb_clear	1	0
format	1	5
format-fail	1	0
format_clear	1	0
ftp-write	4	4
guest	50	0
imap	5	3
ipsweep	9405	6931
land	35	0
loadmodule	3	8
multihop	3	6
neptune	9155	1517488
nmap	2100	257
perl_clear	1	0
perlmagic	4	0
phf	5	0
pod	10133	365
portsweep	1147	9470
rootkit	237	17
satan	180	32452
smurf	5	250128
spy	2	0
warez	1	0
warezclient	439	1327
warezmaster	1	18