Toward Comprehensive Traffic Generation for Online IDS Evaluation

Joel Sommers
University of Wisconsin-Madison
jsommers@cs.wisc.edu

Vinod Yegneswaran
University of Wisconsin-Madison
vinod@cs.wisc.edu

Paul Barford
University of Wisconsin-Madison
pb@cs.wisc.edu

ABSTRACT

We describe a traffic generation framework for conducting online evaluations of network intrusion detection systems over a wide range of realistic conditions. The framework integrates both benign and malicious traffic, enabling generation of IP packet streams with diverse characteristics from the perspective of (*i*) *packet content* (both header and payload), (*ii*) *packet mix* (order of packets in streams) and (*iii*) *packet volume* (arrival rate of packets in streams). We begin by describing a methodology for benign traffic generation that combines payload pools (possibly culled from traces of live traffic) with application-specific automata to generate streams with representative characteristics. Next, we describe a methodology for malicious traffic generation, and techniques for integration with benign traffic to produce a range of realistic workload compositions. We realize our traffic generation framework in a tool we call Trident, and demonstrate its utility through a series of laboratory-based experiments using traces collected from our departmental border router, the DARPA Intrusion Detection Evaluation data sets provided by Lincoln Lab, and a suite of malicious traffic modules that reproduce a broad range of attacks commonly seen in today's networks. Our experiments demonstrate the effects of varying packet content, mix, and volume on the performance of intrusion detection systems.

1. INTRODUCTION

Malicious traffic in the Internet is growing at an alarming rate both in terms of volume and diversity. This escalating threat demands methods and tools to assess the robustness and capabilities of network intrusion detection systems (NIDS) to a wide range of both malicious and benign traffic. Standard methods for NIDS evaluation include the use of canonical packet traces for *offline* tests or traffic generation systems for *online* tests in controlled laboratory settings. Regardless of the approach, the benefits of having established, comprehensive test suites for assessing network intrusion detection systems are obvious. They allow for greater control, reproducibility, and standardized methods for comparing performance of different systems.

The landmark work by McHugh [15] introduced a set of requirements for NIDS test traffic streams. A summary of these requirements is that tests must be conducted with a diverse set of representative packet flows (including packet content) of both benign and malicious traffic. Specifically, the sequences of packets that make up flows in any test must be realistic since NIDS will raise alarms based on signatures of packet exchanges. Likewise, packet headers and payloads must reflect a wide diversity of both benign and malicious content since NIDS also raise alarms based on content signatures.

A natural approach for addressing representativeness in both flows and content is to take empirical traces from real networks for offline analysis. However, this approach is often considered impractical due to standard privacy concerns and the difficulty in accurate labeling of individual packets as benign or malicious. The most notable exceptions are the well known DARPA data sets developed at Lincoln Lab in 1998–1999 for offline NIDS testing [12, 13]. The authors of those studies went to great lengths to create software robots that mimicked user behavior as a means for gathering empirical trace data. While this work has since come under some criticism, as we discuss in § 3, it remains the largest publicly available data set for offline NIDS testing, and has been used in many studies.

Another approach to addressing the challenge of robust NIDS testing is to generate traffic streams synthetically. In principle, this process can result in traces for offline tests or in live streams for online tests. While many traffic generators have been developed for specific network systems tests, none of them address the problem of robust NIDS testing in particular. Perhaps most importantly, the synthetic generation of diverse, representative benign traffic (including payload content) has not been well addressed.

Our goal in this work is to create tools and a test methodology for evaluating the growing number of stateful, protocol-aware intrusion detection systems, with a secondary aim of meeting the test requirements outlined in [15]. These objectives guided our design of a collection of tools, called Trident, which can be used to generate packet traces for traditional offline evaluations, and can also be used in controlled laboratory settings to assess the online performance characteristics of NIDS or other network systems (*e.g.* firewalls). The capabilities of Trident include:

- The ability to generate representative benign traffic streams, including payloads,
- The ability to construct and generate new types of malicious traffic,
- The ability to modulate the mixture of benign and malicious test traffic,
- The ability to modulate the volume of both benign and malicious test traffic,
- The ability to modulate temporal arrival processes of both benign and malicious test traffic.

To the best of our knowledge, no existing toolset provides this combination of capabilities, and we show that they enable a unique and important range of tests for NIDS.

One of the most important features of Trident, and something that distinguishes it from simple malicious traffic generators such as [21], is that it includes representative benign traffic. Trident uses handcrafted automata-based representations of popular network services to generate a wide range of protocol-compliant packet streams. The packets (headers and payload) within the streams are extracted from traces that have been carefully groomed to remove malicious content.

The objective of trace grooming is to create a large pool of packets that is both realistic and diverse. We discuss benefits and drawbacks of three strategies for grooming packet traces for use in Trident. The first approach is to synthetically generate traces, *e.g.*, using statistical models developed from live traces such as was done in the creation of the DARPA data sets. The second approach is to use a NIDS rule set (or sets) to extract benign packets from an empirically collected trace. The third approach is based on our notion of a *trust matrix* which, similar to NIDS rule sets, is used to extract benign packets from empirically collected packet traces. It is important to note that none of these strategies can absolutely guarantee that the resulting benign trace is free of malicious packets. However, our comparisons of these grooming techniques in § 5 treat their capabilities particularly in the case of site-specific NIDS evaluation and tuning.

We demonstrate the capabilities and utility of Trident through a series of tests on live systems in a controlled laboratory environment. We begin by populating Trident with two different benign traffic traces. Next, we create a set of attack modules for malicious traffic commonly seen in today's networks, which we explain in detail in § 6. We use the combination of benign traces and attack modules to assess the behavior of two popular NIDS over a range of traffic volume and packet diversity (content and mix). Our experiments are based on a set of test hypotheses and protocols designed for each system type. The results show that Trident easily exposes an important range of behavior in our test systems. In particular, we show how NIDS performance can be sensitive to the mix of benign application payloads. We also show that the relative proportions of malicious flows to all traffic has a very clear impact on NIDS performance and resulting alarm quality. We further show that while traffic volume has a clear effect on NIDS packet loss, its effect on alarm quality is system dependent. The key implication of our results is that Trident is well suited for evaluating NIDS or other network systems that are protocol-aware and stateful (maintain connection state for detecting anomalous or malicious activity spanning multiple packets or connections). Our results also suggest that Trident would be very useful for tuning NIDS rule sets and the host systems on which they run. Finally, it is also interesting to note that during the course of our tests, a previously unknown UDP fragmentation bug in Bro was discovered which we would not have found without the diverse capabilities of Trident.

In summary, the contributions of this work include the following:

1. Development of a set of commonly seen attack profiles.

2. Development of a framework and integrated set of systems that provide the flexibility to test systems under a range of content, volume and mix of benign and attack traffic.

3. Development of an unbiased trace grooming technique to separate benign from malicious traffic.

4. Demonstration of tool utility through laboratory evaluation of two popular open source NIDS.

5. Collection of scripts for (i) extraction of attacks from the DARPA data set, and (ii) controlled dynamic replay of these attacks (which can be used with other traces as well).

2. RELATED WORK

Several tools exist for generating purely malicious traffic, including [5, 8, 16, 21]. Efforts toward generation of both benign and malicious traffic streams include [10, 19] and a commercial product from Skaion [4]. Trident differs from these systems in its approach to benign traffic generation and the level of flexibility that it provides in controlling the volume, mix and content of produced traffic streams. Trident is also related to the Metasploit and Exploitation Framework projects that provide libraries of common modern attacks [3, 9].

A related study that followed McHugh's critique is described in a paper by Mahoney and Chan [14]. The authors conducted an evaluation of *anomaly-based* NIDS with an enhanced version of the DARPA data set created by injecting benign traffic from a single host (their department web server). Our work, while similar in some respects, differs in several respects: (i) our target systems are much broader than anomaly-based NIDS; (ii) our goal is to provide a flexible and extensible framework for recreating a wide range of attack scenarios by modulating the mix of malicious and benign traffic, control of traffic volumes, and inclusion of representative benign payload contents; (iii) we consider the problem of separating potentially malicious traffic from benign traffic based on protocol knowledge and statistical properties of the traffic instead of relying on a firewall or manual grooming.

Our trust framework for separating attack traffic from a mixed data set is inspired by the work of Jung *et al.* [11]. In that study, the authors proposed a hypothesis testing framework for detecting malicious scanners. We use a similar though more simple method in this study. While beyond the scope of this work, a hypothesis testing framework could be incorporated into our benign trace grooming algorithm. Similarly, Antonatos *et al.* examined the problem of generating benign traffic based on statistical properties of existing traces [7]. The details of their algorithm differ from ours in that their effort was limited to replicating payload contents while our environment is broader, allowing control of not only packet content but also higher level attributes such as flow arrivals and source address distribution.

3. MCHUGH'S CRITIQUE OF LINCOLN LAB IDS EVALUATION

A starting point for our work is the comprehensive critique by McHugh of the DARPA-sponsored 1998–1999 IDS evaluations [15]. A key contribution of McHugh's work is not only that it identified the shortcomings of the reported evaluations but it used these observations as a springboard for specifying requirements for comprehensive IDS testing.

The data sets used in Lincoln Lab evaluations consist of a 5 week-long trace of packet data and feature 58 distinct attack types. Weeks 1 and 3 in the trace contain entirely benign traffic. Week 2 contains benign traffic mixed with labeled attack traffic that can be used to train anomaly-based intrusion detection systems. Weeks 4 and 5 contain test data which has an unlabeled mix of attack and benign traffic. Despite its problems, it remains one of the largest and most comprehensive data sets for IDS evaluation available today, and is suitable for offline tests. While online tests using a trace replay tool such as [1] are certainly possible, these tools do not allow for flexible manipulation of the resulting traffic streams in the ways described below.

We distill the major points of McHugh's evaluation below, and use these as motivation for our IDS traffic generation architecture.

Benign Data. The Lincoln Lab evaluation did not validate the false alarm characteristics of the background (benign) data. In par-

Table 1: Summary of DARPA and CSL data sets.

Dataset	TCP			UDP			ICMP	
	Pkts (M)	Bytes (GB)	Flows (K)	Pkts (M)	Bytes (MB)	Flows (K)	Pkts (K)	Bytes (MB)
Week 1	14.1	2.85	456	0.69	64	18	12	4
Week 2	12.4	2.25	471	0.72	66	27	47	3
Week 3	14.4	3.10	486	1.40	155	14	10	0.6
Week 4	12.1	2.04	333	1.98	214	21	93	27
Week 5	20.6	4.43	864	3.65	346	23	37	12
CSL	12.4	11.04	38	2.24	421	178	92	9

Table 2: Port distribution of CSL Traffic.

Port	% No. Flows	% No. Pkts
123/udp (NTP)	38.40	2.64
53/udp (DNS)	34.92	2.06
25/tcp (SMTP)	4.72	0.26
80/tcp (HTTP)	2.72	3.55
3124/tcp	1.36	0.18
23127/udp	1.02	0.16
9618/udp (Condor)	0.95	0.14
3126/tcp (.NET)	0.94	0.07
21/tcp (FTP)	0.65	0.12
22/tcp (SSH)	0.02	2.67

Table 3: Port distribution of Darpa Week 1 Traffic.

Port	% No. Flows	% No. Pkts
80/tcp (HTTP)	88.14	16.41
25/tcp (SMTP)	4.79	2.17
53/udp (DNS)	3.10	1.31
123/udp (NTP)	0.70	0.46
23/tcp (TELNET)	0.49	26.95
79/tcp (FINGER)	0.42	0.08
21/tcp (FTP)	0.23	0.35
110/tcp (POP3)	0.09	0.03
37/tcp (TIME)	0.08	0.01
22/tcp (SSH)	0.03	4.54

ticular, there was no rationale provided to convince the reader that the observed *rates* from this data set would be similar to those observed in live environments. Moreover, since *volume* and *burstiness* (in terms of packet arrival characteristics) of background traffic vary widely across networks, it might be impossible to create a single representative background trace. Packet storms resulting from misconfigurations that are common in real networks and often resemble flooding attacks were ignored. In summary rates, volume, and burstiness of benign traffic are important considerations, and timing parameters derived from network emulators must be validated.

Malicious Data. The Lincoln Lab evaluations did not attempt to ensure that the *mix* of malicious and benign traffic in the trace data was realistic. Second, the number of systems that were subject to attacks was quite limited and no attempt was made to validate that this distribution was realistic. Third, the *attacker-centric* approach used by the evaluators could produce biased results and offers little toward understanding IDS behavior. Finally, McHugh notes that as significant effort is expended by the research community to generate attacks for testing intrusion detection systems, an “attack-on-demand” facility that tracks and replicates the latest attacks would be extremely useful to the community. In summary, benign/malicious traffic mix is an important consideration as is the diversity and current prevalence of malicious traffic used in tests.

Finally while simple topologies are commonly used to conduct experiments and build data sets, the burden rests on the experimenters to prove that the artificial environment does not significantly alter the meaningfulness of the experiments.

4. DATA COLLECTION

The data sets used to derive benign packet traces for this study include the five week-long traces from the 1999 DARPA data set, and a trace (headers and payloads) collected for 100 minutes (around 100 GB) from our departmental border router, which we refer to as the CSL trace. Weeks 1 and 3 of the DARPA data set contain no malicious traffic and week 2 includes labeled malicious training data while weeks 4 and 5 include malicious test data.

Table 1 provides a summary of both data sets. There are several contrasts between the two data sets. First, although the TCP byte counts for the CSL trace are larger than that of a typical week of the DARPA data, the number of flows is an order of magnitude smaller. This suggests that large data transfers are more prevalent in the CSL network than in the DARPA trace. Second, the large number of UDP flows seen in the CSL data is dominated by NTP and DNS traffic. The high volume of NTP traffic seen in the CSL network is partly attributed to [18]. Third, comparing the port distributions in Tables 2 and 3, we see that the application mix (as identified by port numbers) in each trace is also quite different. This difference is to be expected given the fact that the CSL data was collected in January, 2005 and the DARPA data was generated in 1999. It also underscores the need for continuous collection and updating of test data.

5. CONSTRUCTING A BENIGN TRACE

One of the most important aspects of NIDS evaluation is a thorough assessment of the system’s propensity to generate alarms in the absence of malicious traffic (false positives). The quantity of false positives is intrinsically tied to both the NIDS under test and the nature of benign traffic in the test environment. Therefore, one of the essential aspects of NIDS evaluation for any network is a benign traffic workload that features the spectrum of characteristics that are *typical* or *expected* for that network. While one might be able to readily capture a collection of packet traces from the network over an appropriate period of time, the difficulty arises from the fact that we expect these traces to contain a mixture of both benign and malicious traffic. So, the question becomes *how to identify and isolate the benign traffic*. In many ways, this is exactly the intrusion detection problem.

There are several possibilities for populating Trident with benign traffic payloads. We discuss three strategies that might be employed as a way to explore the design space. While other techniques may be possible, we believe that the strategies we discuss are reasonable, effective and cover a large portion of the design space.

- ***NIDS-based Strategies:*** The first strategy is to use a well-known NIDS such as Snort (that is well known to generate a lot of false alarms but also detects attacks accurately) to groom a packet trace taken at a local site. We argue that this is a highly problematic approach since a portion of the connections that are removed are likely to be those that are of "highest interest" in the sense that they are benign packets that trigger alarms (false positives).
- ***Synthetic Generation Strategies:*** A second strategy is to use synthetic traffic generated using software robots that emulate user behavior. The idea is to craft the robot to ensure that it only creates connections with known good hosts (either local or remote). This data is then used as a basis for further expansion of the trace through synthetic generation of packets (as in the DARPA data set). While this strategy is clearly limited in terms of representativeness from an application mix and payload perspective, it may be appropriate for certain environments. A further benefit of this method is that since the base trace is generated by robots, it may enable trace data sets to be shared.
- ***Trust-based Strategies:*** The third strategy is to groom a packet trace taken at a local site using connection heuristics (*e.g.*, failure rates or scanning characteristics). This approach exploits the dif-

ferences in connection characteristics of benign versus malicious sources based on a model of malicious connection behavior. This technique is attractive because it is based on transport level characteristics, does not require knowledge of application semantics, and is not biased by a particular system (NIDS independence). We posit that a trust-based grooming strategy results in a set of packets labeled as benign that have a higher opportunity for uncovering false positive behavior. It is, however, limited in that it might miss targeted attacks by sophisticated adversaries that have connection characteristics sufficiently similar to benign users.

The remainder of this section explores the trust-based strategy. We begin by defining the specifics of our trust-based grooming methodology. Next, we evaluate its performance using the CSL packet trace. We conclude the section by discussing some of the strengths and pitfalls of this strategy.

5.1 Trust-Oriented Grooming

Our trust-based approach addresses the problem of separating benign traffic from a diverse trace by developing a systematic set of rules for attributing each connection in a trace with a specific level of trust. We begin by defining a framework for establishing the trust assignment rules that is (*i*) *not NIDS-specific* and (*ii*) *not application-specific*. Thus, we consider the framework to be *unbiased*. Within this framework, our approach is to associate trust levels with the characteristics and observed behavior of the participant end hosts and the networks in which they reside. In this sense, trust assignment becomes site specific. We believe that site-specificity is essential for effective NIDS testing, but may not directly lend itself to standardized benchmarking.

Our framework supports attributing trust to each host at multiple granularities. We begin by considering individual hosts as being *trusted*, *neutral* or *suspect*. This estimate is derived simply based on connection utility and endpoint location (local vs remote). We then extend this notion to define *trusted, neutral* and *suspect* networks. During the execution of the trust assignment algorithm, these labels are computed based on the behavior of the individual hosts in each network. However, certain networks can also be pre-identified as trusted or not trusted and thus all of the hosts in those networks will be either whitelisted or blacklisted. For example, a specific network may be known (based on personal knowledge) to be well managed, and known to have strong, enforced security policies (*e.g.,* one's own network). Likewise, certain networks are known to be a common source of malicious traffic and might be blacklisted as a suspect network.

5.2 The Trust Matrix Algorithm

Our trust assignment algorithm, which we call the *trust matrix* (or TM), uses three basic metrics for its decisions, including (*i*) *endpoint location*, (*ii*) *number of failed/inbound/outbound connections*, and (*iii*) *volume of traffic exchanged*. Clearly, this framework for benign traffic grooming can and should be modified by individuals for their own environment and based on their own experience. The algorithm that we derived from this framework for isolating benign traffic assumes *transitivity of trust* and has the goal of identifying hosts as either trusted or suspect as follows:

Step 1. Trust local and white listed hosts. We assume that the local network is well managed and that all its hosts are most trustworthy. We also assume there is a possibility that other networks are sufficiently trustworthy based on knowledge of management and security practices. Label all hosts in these networks as trusted[1].

Step 2. Trust remote servers. Since we trust local hosts, we also trust connections that they initiate [2]. It then follows that we can trust the remote servers to which they connect—label these hosts as trusted.

Step 3. Distrust scanners. We identify sources that have failed connections sent to multiple ports or destinations as suspect[3]. We use a simple heuristic to identify scanners that considers the ratio of good connections (connections that are established and successfully receive data) to all connections. We call this the host's β value. The threshold parameter β_H determines the minimum acceptable value of β for a trusted host and β_L determines the minimum value for a neutral host. Hosts with β values below β_L are labeled suspect. *An important point is that this heuristic is effective in capturing not just port-scanners, but also describes most worms and any malware that randomly selects IP addresses for propagation*.

Step 4. Trust well behaved clients. We identify remote clients that have high β values (above β_H) and label those hosts as trusted.

Step 5. Distrust hosts in suspect networks. Identify otherwise unlabeled hosts in networks with scanners as network suspect hosts. If the ratio of unsuccessful connections to all connections from this network falls below β_L label the all unlabeled hosts as suspect.

Step 6. Trust peer networks. Classify networks that exchange high volumes of data with the local network as peer networks. Label all unlabeled hosts in peer networks as trusted.

Step 7. Trust well-behaved networks. Identify networks with high β values above β_H and label all unlabeled hosts in these networks as trusted.

Step 8. Label remaining unlabeled hosts as trust neutral.

There are two aspects of this process that are noteworthy. The first is that it is designed to be *conservative, i.e.,* it favors labeling hosts as suspect. A host is deemed suspect if the sum of its activities fail to meet any single trust metric. Second, the algorithm above is ad hoc and is meant to be modified and enhanced on a per network basis. The TM is summarized in Table 4 and we demonstrate its utility on a live traffic trace in § 5.4.

Table 4: Ideal host classification in the trust matrix (T = trusted, N = neutral, S = suspect, X = not applicable).

	Hosts			
	Local	Remote		
		Server	Client	
			Neutral	Scanner
Trusted Network	T	T	T	S
Neutral Network	X	T	N	S
Peer Network	X	T	T	S
Suspect Network	X	T	N	S

5.3 Estimating Trust Matrix Parameters

We now describe how an empirical packet trace can used to estimate reasonable values for the threshold parameters β_H and β_L. When individual IP addresses are considered, β is simply the ratio of successful connections to all connection attempts with an implicit β value of 1 for otherwise trusted hosts such as local clients. For network aggregates, we use the ratio of all connections from a given network to determine the network's β value. Figure 1 shows the distribution of β values for individual hosts and two network

[1]While this assumption is reasonable for a well-managed network such as ours, it may be unsuitable for other networks. Other considerations might include prevalence of spyware or otherwise compromised hosts among local clients.

[2]This does not include "follow-up" connections. This situation occurs when a remote client first initiates a connection to a local host, which leads to a follow-up connection from the local host.

[3]This technique is related to the methods described in [11].

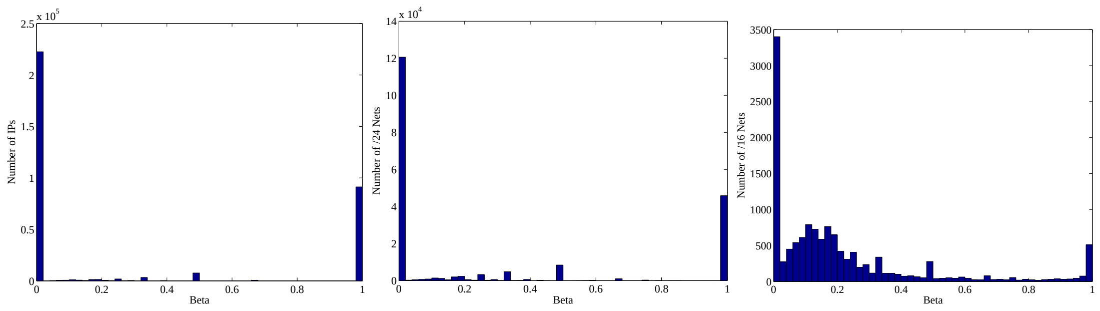


Figure 1: Distribution of β values in networks of various sizes from the CSL trace (individual IP addresses (left), /24 networks (center), /16 networks (right)).

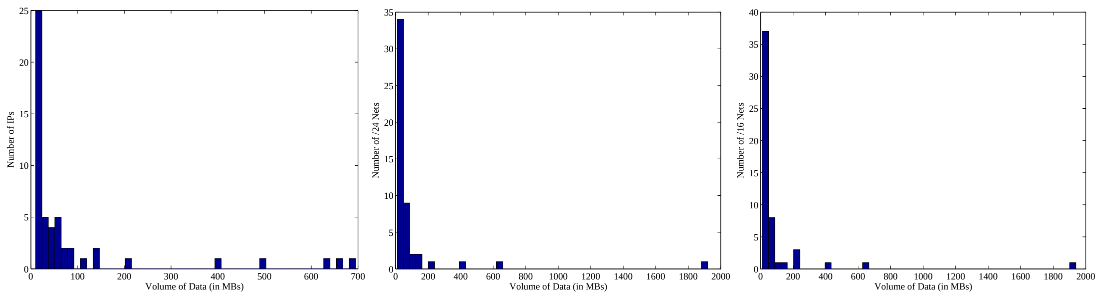


Figure 2: Summary of transfer volumes from the CSL trace for individual IP addressees (left), /24 networks (center) and /16 networks (right).

aggregates using the CSL trace. The first observation from these plots is that the distribution of β at the individual host IP level is bimodal, *i.e.*, hosts tend to be either well-behaved or have many failed connections. This effect has an important impact on our ability to identify malicious scanners. Second, the results show that the desirable bimodal-like distributional properties of β continue to hold at /24 network aggregates but are not as strong for /16 aggregates. This suggests that a good starting point for considering network trust in this trace lies somewhere between /24 and /16 aggregates. These results provide the basis for fixing our β_H value at 0.8 and β_L value at 0.2.

In Figure 2, we consider the parameters for identifying peer-networks, which are defined as networks that participate in high-volume data transfers with a trusted local network. The histograms for the CSL trace suggest that there are few networks that exchange large volumes of data; these are easily isolated. The figure shows that a volume of 100 MB appears to be a reasonable threshold for separating high-volume transfers, and this value is not overly sensitive to the size of the network. We fix the volume threshold parameter at 100 MB for the CSL trace.

We also explored the dynamics of the trust assignment algorithm as the network radius expands (where radius 0 = /32 network, radius 1 = /31, etc.). Our goal was to maximize the number of accurately labeled hosts (trusted or suspect) while minimizing the number of inconsistent labels (*i.e.*, trusted hosts in suspect networks or suspect hosts within trusted networks). The results reinforce the results from the earlier two heuristics for optimal network aggregation for trust labeling, and suggest that the correct aggregation for these traces lies in the range /24 to /20. This result is likely related to the administrative granularities used in allocation of IPv4 addresses.

5.4 Trust Matrix Evaluation

Evaluation of the resiliency of the TM to false negatives involves validation with a NIDS rule-set and conducting a manual examination of the trace for each missed alert by the TM. More specifically, how often are legitimate hosts marked suspect (false positives) and how often are malicious hosts marked trusted by the TM algorithm (false negatives)? We begin with an analysis of the former using the CSL trace. While we do not have full “ground truth” for this data, we approximate it through a manual tagging process. We first classify sources marked suspicious by the TM by protocol (port numbers). We then examine activity of several exemplars from each pool to determine whether the sources are malicious or benign. In our experience, the dominant pools tend to be either entirely legitimate, malicious or misconfiguration. However, there are some less popular ports, often with a single source, where we simply do not know enough about the activity to label it accurately. Similarly, we encounter activity that does not include sufficient context to determine the intent of the source (*e.g.*, ICMP echo requests are used for both benign measurement and by worms such as Welchia). In all, a small fraction (4,188—little over 1%) of the 373,395 sources in the full trace were deemed suspect.

We summarize our evaluation of the trace in Table 5. First, we find a high degree of failed connections directed at our departmental mail servers. An obvious explanation for this is spammers looking for open-relays. Second, we find that certain legitimate DNS requests are dropped by the DNS server. This is the expected behavior of BIND when requests timeout. Likewise, a small fraction of NTP requests seem to be dropped by the NTP server though we do not have an explanation for this behavior. The TM also marks certain sources running Condor[4] as suspicious because these sources can send periodic one-way UDP updates to a server (*i.e.* they do not receive an acknowledgment). This incorrect labeling illustrates a current weakness of our TM. While TCP connection semantics

[4]Condor is a widely-used computational resource sharing system.

Table 5: Activity summary of sources deemed suspect by the TM. Sources under category "Legitimate" are considered false alarms. "Other" includes malicious, legitimate, and unknown.

Service	*Port*	*No. Sources*	*Category*
SMTP	25	940	Spam (Open-Relay)
DNS	53	851	Legitimate
NTP	123	469	Legitimate
Condor	9618	322	Legitimate
DCERPC	1025	230	Exploit
Beagle	2745	127	Exploit
HTTP	80	107	Worms, Open-Proxy
MyDoom	3127	87	Exploit
EDonkey	4662	71	Misconfiguration
FTP	21	54	Legitimate
MS-SQL	1433, 1434	10	MS-SQL Probe Response,Slammer
Other		960	—

can often be extended to most UDP sessions, some isolated applications do not follow a two-way data exchange and are not presently accounted for in the TM so may need to be whitelisted. The TM also successfully detects several instances of malicious traffic such as sources trying to exploit DCERPC vulnerabilities on port 1025, sources scanning for Beagle and MyDoom backdoors, MS-SQL password probing attempts and MS-SQL Slammer worm.

A robust evaluation of the resiliency of the TM to false negatives involves validation with respect to a NIDS rule-set and conducting a manual examination of the trace for each missed alert by our system. This task can be particularly labor intensive considering the size of the data set and volume of false positives in modern NIDS rulesets as we discuss below. Instead, we look for false negatives with respect to the most common background radiation signatures developed for the Bro IDS [17]. Our results revealed a *zero false-negative rate* [5].

Next, we compare the TM groomed trace with the result of alerts generated by Snort. We found that running Snort (version 2.4.3) with a default configuration and snort-rules-pr-2.4 raises alerts on 3,367 distinct sources. This result is in line with the number of sources flagged as suspect by the TM. We summarize the dominant Snort alert categories in Table 5. Examining behavior of typical sources from these dominant categories suggests that they all correspond to legitimate traffic. However, several attack sources were also detected by Snort's payload signatures (SQL-probing, SQL-Slammer) and its portscan detector (DCERPC, Beagle). While Snort detected all of the MS-SQL attacks, it only detected a small portion of other obvious background-radiation attack instances such as 8/230 DCERPC attacks, 11/127 Beagle attacks.

In comparing the sources flagged as malicious by both Snort and the TM, we find that there is less than a 10% overlap (308 sources). We believe that this highlights the potential utility of the TM approach—a large fraction of packets that trigger false positives in Snort are not flagged by the TM. The overlap includes all of the background-radiation scanners detected by Snort, such as the DCERPC exploit and Beagle backdoor attempts, *i.e.*, sources flagged by the TM are a superset of Snort sources for these attacks. Some of the legitimate DNS and NTP sources flagged as malicious by the TM also confuse Snort's portscan detector. For these services, the Snort false positives are a subset of the TM's false positives listed in Table 5.

5.5 Limitations of the Trust Assignment Algorithm

While the TM has the advantages of being unbiased and simple,

[5]While we detected several instances of true alerts such as MS-SQL Slammer, no background radiation attacks were missed.

Table 6: Summary of dominant Snort false alert categories. Note that individual sources could contribute to multiple Snort alerts.

Alert	*No. Sources*	*No. Instances*
ICMP PING NMAP	877	1327
WEB-MISC /doc/ access	867	2027
ICMP Large ICMP Packet	322	13,021
WEB-MISC Invalid HTTP Version String	219	1162
(portscan) UDP Portsweep	166	570
FTP command overflow attempt	64	5690
WEB-MISC robots.txt access	60	93
(http_inspect) DOUBLE DECODING ATTACK	58	191

it also has limitations. It is likely that a limited amount of malicious packets will persist after trace grooming. For example, a randomly scanning HTTP worm may send exactly one connection request to the local network and that request may contact an open web server. Detecting the presence of such attacks is impossible using our trust assignment algorithm. While this connection might be flagged by an IDS, we reject the approach of using IDS rule sets for trace grooming since loose rule sets might flag benign traffic that would otherwise cause false positives in tests. We argue that it is better to have some unexpected true positives in test results than to reduce or eliminate the possibility of seeing false positives due to over-sanitized benign traces. Second, we note that without inspecting all packets of a trace (a task we consider to be infeasible), there is a possibility of false negatives during tests. The potential difficulty this poses for analyzing test results may be an inevitable trade-off of having the realism of traffic streams collected from an operational network. Third, certain site-specific applications such as Condor that use one-way UDP streams might need to be whitelisted since they do not have a well-formed notion of "goodness". Extending our methodology to consider this traffic is an area for future work. Finally, our notion of trust depends on the perspective of the local network. That is, a network perceived as hostile by us might be a neutral network for another service provider with whom the hostile network has a peering relationship.

6. THE TRIDENT SYSTEM

In this section we describe how application-specific benign traffic streams and malicious traffic streams are generated within the Trident framework.

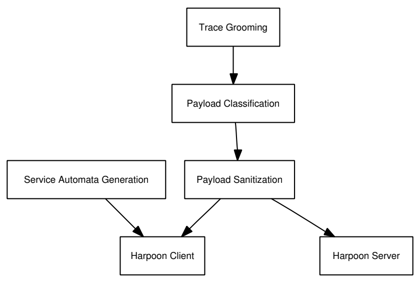


Figure 3: Steps for benign traffic generation in Trident.

6.1 Benign Traffic Generation

Several studies of IDS performance, including many papers that use the DARPA data set, consider detection characteristics from an off line evaluation (*e.g.*, [6]). As we show in the results of our laboratory experiments, the dynamic characteristics of traffic streams, notably the mix and volume, can have huge impact on NIDS performance and provide key insights for system tuning. Therefore, one of the challenges of benign traffic generation is to dynamically generate diverse traffic streams based on knowledge obtained from a limited set of traces. To this end, we considered three strategies for generating benign traffic.

The first strategy is to use a tool like `flow-replay` [1] to create flows without any modification to packet headers or payloads. The most significant drawback of this approach is that it only recreates the client side of a connection. A component of Trident called `attack-replay` (described below) provides the capability to replay both endpoints of a flow. While this strategy is simple, it is clearly limited in the amount of diversity that can introduced into a traffic stream.

A second strategy is statistical replication of payloads for protocols based on byte-level properties. To our knowledge there has been only limited work in this area. For example, in the context of anomaly detection, language-independent statistical profiles such as n-grams have been proposed to model application payloads [23]. These techniques are attractive because they allow traffic generation systems to be oblivious to session/application layer protocols. However, the current trend toward protocol-awareness in modern NIDS and traffic analyzers suggests that traffic generation needs to be application-aware to generate meaningful headers/payloads that exercise the NIDS in ways similar to real traffic.

Another approach, and the one we adopt in Trident, is protocol-aware emulation based on payload interleaving. Payload interleaving is our term for dynamic construction of flows through random selection of packets from payload pools corresponding to particular states in a service automaton, as we describe below. This method supports the generation of synthetic traffic streams with realistic application level headers and payloads. While not a focus of our evaluation, the statistical techniques discussed in § 5 should be considered complementary and could be used to generate application level data, for example the entity body of an HTTP connection. We describe the components of our protocol-aware emulation scheme below.

• **Automata Generation.** At the heart of our benign traffic generation system is a collection of automata with states that describe classes of packets observed in a specific service. In our preliminary exploration of the feasibility of this approach, we use basic automata to describe the most popular services seen in the CSL trace and in the DARPA data set. We do not claim completeness of these automata or suggest that they exercise all classes of NIDS. We use them as examples to demonstrate the utility of our methods and to show how they can accommodate flexible recreation of a broad class of protocols.

Our automata describe each service through a three phase abstraction that is typical of most network protocols. The first stage, *prologue*, describes the application-level client server handshake. The next stage is *dialog*, in which client and server exchange data. The final stage is *epilogue*, in which the participants agree to gracefully tear down the connection. Each stage in the conversation could involve several states in the automata and the final stage is optional in some protocols such as HTTP. We created automata that model the packet exchange protocols for HTTP, SMTP, DNS, Telnet, FTP and SSH (*i.e.*,the most popular services from our data sets). Our automata-based abstractions for HTTP and SMTP are shown in Figures 4(a) and 4(b). Pipelined HTTP requests are currently not supported but should be an easy extension.

A weakness of a protocol-aware automata-based system such as ours is that the effectiveness of the evaluation is related to the quality of the automata. It is our hope that a library of automata will be developed by the research community over time.

• **Payload Classification.** The raw traces classified as benign (*i.e.,* trusted or trust neutral) are given as input to the payload classification module which we call `payload-gen`. The purpose of `payload-gen` is to classify packets in the trace into various pools that correspond to particular states of different service automata. In this step, packets generated in the same application state, but from different flows, are aggregated into the same pool. This aggregation does not preserve packet ordering from any individual flow.

• **Payload Sanitization.** Following classification, payloads are discarded or modified to ensure that they do not violate a simple set of requirements. Discard is appropriate if the original payload suffered truncation during packet capture or the payload does not match a valid automata state. Modification is generally done to simplify service automata definition and processing, and to avoid generation of false alarms that would result simply as an effect of interleaving. Our current approach is to be aggressive in these normalization steps. For example, with HTTP, we remove `Connection`, `Content-length`, and `Transfer-encoding` headers from server responses, since a NIDS could conceivably use these items to monitor a connection in progress. Since we wish to arbitrarily use client and server payloads without maintaining elaborate state or dynamically rewriting payloads, we remove the echoed address from the server response since it is not required for correct protocol operation. An effect of our sanitization is that we may underreport the levels of false alarms generated by the NIDS that we evaluate.

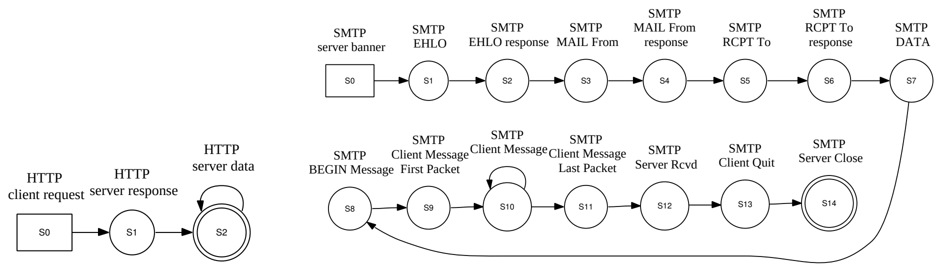
• **Content Aware Traffic Generation via Harpoon.** We wrote a new traffic generation plug-in for Harpoon [20] to execute the application state machines and transmit sanitized payloads. Control of state machine processing is done on Harpoon clients. Harpoon servers simply respond to requests to send a certain number of packets from specified application payload pools.

In addition to the distributional data used in Harpoon's default TCP traffic generator, we define $P_{NumPackets}$ and $P_{FlowSize}$ distributions. As prologue and epilogue state machine stages are executed, one packet is sent from the appropriate application payload pool. For the dialog stage, the number of packets to send in each state is chosen from a distribution $P_{NumPackets}$. In addition, the dialog stage ends when the total traffic sent exceeds a value chosen from a distribution $P_{FlowSize}$. Connections are initiated according to a distribution $P_{InterConnection}$. Each of these distributions may be specified for each service. The effect of modulating the flow size, interconnection time, and active session distributions is that the overall traffic volume and application mix can be tuned as desired.

The exchange of application-layer payloads according to a given service automaton is done using standard user-level sockets, *i.e.*, above the transport layer. Benign traffic streams produced by Trident are therefore targeted at intrusion detection systems focused on layers above the transport layer. Presently, we do not explicitly account for non-malicious transport-layer anomalies that may have been present in a trace captured in a live network, such as misconfigurations or implementation bugs. Our malicious traffic generators, described next, include exploits that target both network and transport layers, as well as higher layers.

6.2 Attack Traffic Generation

• **General attack traffic creation.** MACE is a modular attack composition framework that consists of three primary components: (i) exploit, (ii) obfuscation, and (iii) propagation, as well as a number of functions to support interpretation, execution, and exception handling of attack profiles. For this work, we extended the existing set of exploits in MACE from 5 to 21 attacks [21] and enhanced its ability to modulate attack volumes. A taxonomy of available



(a) HTTP service automaton.

(b) SMTP service automaton.

Figure 4: Service automata for two common application protocols.

Table 7: Taxonomy of MACE Exploits.

Host Based					**Network Based**
APPLICATION LEVEL			TRANSPORT LEVEL		
Worms	Back-doors	DoS	Fragmen-tation	Other DoS	
Welchia			rose		
Nimda	mydoom		teardrop1	synflood	smurf
CodeRed2	sdbot	winnuke	teardrop2	pod	fraggle
Blaster			bonk	land	
Dameware			nestea	jolt	
Sasser			oshare		

Table 8: Summary of Trident tools developed for NIDS performance evaluation.

Name	**Description**
attack-replay	A flow replay tool that allows two-way replay of a packet trace.
autom-gen	A script that stores service descriptions and generates service-specific automata for Harpoon.
exec-groom	A traffic grooming algorithm that uses trust heuristics to separate benign traffic from suspicious traffic.
payload-gen	A tool that reads a groomed packet trace and outputs packet pools that correspond to automata states.
payload-sanitize	A tool that sanitizes inconsistencies in protocol headers that are introduced due to interleaving.
split-darpa	A script to separate malicious DARPA traffic from benign based on labels.
harpoon plugin	A traffic generation plugin for Harpoon that executes the service description automata to produce application payload traffic.

MACE exploits is provided in Table 7. Our objective is not to provide a complete attack database for intrusion testing, but to provide a spectrum of attacks that exercise NIDS in sufficiently diverse ways and to support a set of basic building blocks that can be used to create additional (and perhaps as yet unseen) attack vectors.

• **DARPA attack recreation.** The DARPA data set provides a collection of 58 different attack instances. To extend the utility of Trident, we added the capability to dynamically replay these attacks. We began by developing a tool called `split-darpa` to distill labeled attacks from the mixed traces. Due to possible inaccuracies in the labeling, `split-darpa` was able to automatically isolate 56/58 attacks in the data set. Next, we developed the ability to perform dynamic replay of attack traces with the tool `attack-replay`. One of the key aspects of this effort is that for TCP attacks, reassembly of payloads is done before sending the packets through TCP sockets. All state is maintained at the client and appropriate server responses are fed to the server through an out-of-band control channel in a timely manner. For UDP and ICMP packets, the traffic is transmitted through raw sockets.

6.3 Test Methodology

The objective of laboratory-based experiments reported in § 7 is to demonstrate the utility of Trident by evaluating the effectiveness of specific NIDS configurations along dimensions of packet diversity (content and mix) and traffic volume.

Test Setup. The NIDS we evaluated in our experiments were Bro (version 0.9a8) and Snort (version 2.3.0). For Bro, we used the default `brolite.bro` policy, and for Snort, we used the default `snort.conf`. We included a third NIDS configuration consisting of Snort (version 2.3.0) with a recent snapshot of signatures from Bleeding Snort [2]. Each NIDS ran on a separate workstation with a 2 GHz Intel Pentium 4 processor, 1 GB of RAM, and Intel/PRO 1000 network cards. FreeBSD 5.1 was installed on each machine[6].

The three NIDS hosts were connected to a Cisco 6500 enterprise switch/router. Harpoon, MACE, and attack-replay traffic generators were also connected to this switch, which was configured in such a way that the three NIDS received all traffic sent between the traffic generation hosts. We used two large (2^{16}) address spaces as "internal" and "external" networks, configuring interface aliases on each traffic generation host.

We ran three sets of experiments. The first set of experiments was designed to establish a baseline of alarm behavior for each NIDS. We first generated a low (5 Mb/s) rate of benign traffic from the CSL and DARPA data sets, tracing all traffic. We then used the captured packet trace in an offline manner to produce a baseline set of alarms generated by the three NIDS configurations. Similarly, for each exploit produced by MACE and each exploit from the DARPA data set, we generated a baseline set of alarms for the three NIDS configurations.

In the second set of experiments, we altered the mix of flow volumes between benign traffic generated by Harpoon, and malicious traffic generated by MACE (CSL) or attack-replay (DARPA). To effect different mixes, we kept the benign traffic level constant at about 20 Mb/s, while introducing different levels of malicious flows. The specific mixes we used were 100% benign flows, 90% benign flows, 50% benign flows, and 10% benign flows. Below, we refer to these test setups as mix100, mix90/10, mix50/50, and mix10/90, respectively.

In the third set of experiments, we used three different levels of traffic volumes. We tuned Harpoon to generate roughly 20 Mb/s, 40 Mb/s, or 60 Mb/s for each of the CSL and DARPA data sets. For each traffic volume level, we tuned MACE or attack-replay to produce approximately 2 Mb/s, 4 Mb/s, and 6 Mb/s of aggregate

[6]On each host we modified the kernel parameters `debug.bpf_bufsiz` to 4194304 and `debug.bpf_maxbufsize` to 8388608 as suggested in the Bro documentation. Snort presumably can benefit from this change as well so we applied the change to each NIDS host.

attack traffic, respectively. Below, we refer to these tests exploring the effect of traffic volumes as vol20, vol40, and vol60.

We ran each experiment for 15 minutes. On each NIDS host, we measured CPU and memory usage every 5 seconds using `vmstat`. We also took note of packet drops reported by each NIDS upon shutdown[7].

Evaluating Results. Using the results of running the malicious traffic in the baseline experiments, we constructed representations of the types and number of alarms at each NIDS we expected to observe for each exploit. For the second and third sets of experiments, we recorded the number of times an individual exploit was executed by MACE or attack-replay. We then were able to compare the alarms produced by each NIDS with what we would expect, given the specific exploits launched over the duration of the test. We wrote a script to automatically process this representation of expected alarms along with the actual log files produced by a NIDS during a given test. The script reported the set of alarms produced by the NIDS, along with a frequency of occurrence, and the expected number of occurrences. We used these counts to generate relative alarm efficiency and effectiveness values [22]. Efficiency is defined as $Efficiency = \frac{TruePositives}{AllAlarms}$, and is a measure of false positive occurrence, where a value of 1 means that there are no false positives and a value of 0 means that all alarms are false positives. Effectiveness is defined as $Effectiveness = \frac{TruePositives}{AllPositives}$ is a measure of false negatives, where a value of 1 means there are no false negatives (*i.e.*, all alarms that should have occurred did occur).

7. LABORATORY-BASED IDS EVALUTION

We illustrate the utility of our framework through results of an experimental evaluation of Bro, Snort and Bleeding Snort performance under varying traffic content, mix and volume. While these results substantiate the effectiveness of the measurement tools and the potential of the methodology, they should be interpreted with the following two caveats:

1. The results are limited by the representativeness and diversity of our protocol automata.

2. Our goal is to conduct black-box evaluation of NIDS performance. So we do not perform in-depth analysis of behavioral causalities.

As a result, *these results are not intended to be used as a head-to-head comparison of the systems or their rulesets*. However they are valuable in that they demonstrate effects of varying benign and malicious traffic content, mix and volume, and establish the feasibility of our approach.

Baselining Benign Traffic / Evaluating effect of interleaving. An important question is how payload interleaving, the process of random selection of packets from individual payload pools based on the states in each service automaton, affects alarm characteristics. In particular, it is important to demonstrate that no legitimate alarms are introduced due to data consistency issues[8].

Table 9 shows the number of *unique* alarms produced by the three NIDS setups in both offline and online configurations using the CSL and DARPA data sets. For the offline setup, we ran each NIDS configuration using each trace after grooming, but prior to payload classification and sanitization so that the original flows (including IP and TCP headers) were left intact. In the online setup, we used Trident to generate flows using the groomed, classified, and sanitized traces. We see that the number of unique alarms is consistently less for the online test. This effect is caused by the conservative nature of our sanitization process and by the fact that our laboratory tests are run in a relatively simple environment. Furthermore, the set of alarm types generated in the online tests is a subset of the alarm types produced in the offline setup. Alarms unique to the offline tests are most often related to transport, addressing, and routing (*i.e.*, layers 3 and 4) and the common alarms are application-related. For example, in the offline CSL test, Bro and both Snort variants report address and port scan activity, as well as small packet fragments that may indicate nefarious activity. Since Bro maintains connection state, it also reports unexpected transport layer behavior, such as odd TCP window resizing, TCP checksum errors, and potential split routing. Since we randomly traversed the laboratory address spaces in our online tests, it is not very likely that scanning alarms will be triggered (unless the configuration is set to cause such alarms). Examples of Snort alarms most common to the offline and online tests include HTTP URLs associated with malicious activity (*e.g.*, certain PHP scripts) and an FTP CWD with the directory "…" (rather than ".."). Examples of Bro alarms most common to the offline and online tests include detection of the FTP PASV command and detecting a single carriage return at the end of HTTP or SMTP commands, where there should be a carriage return-line feed pair.

Table 9: Number of unique alarms generated for each data set for offline and online setups. The offline setup uses groomed traces with the original flows (IP and TCP headers) left intact. The online setup uses Trident to generate flows using the groomed, classified, and sanitized traces.

	CSL		DARPA	
IDS	**Offline**	**Online**	**Offline**	**Online**
Bro	64	9	21	11
Snort	19	11	47	21
Bleeding Snort	20	17	60	24

Table 10: Alarm counts for Bro, Snort, and Bleeding Snort for a single instance of each MACE exploit.

exploit	Bro	Snort	Bleed Snort	exploit	Bro	Snort	Bleed Snort
SYN flood	0	0	0	**oshare**	2	1	1
blaster	0	1	1	**pingofdeath**	4	0	0
bonk	3	1	1	**rose**	2	0	0
codered2	2	4	3	**sasser**	0	0	11
dameware	0	0	0	**sdbot**	0	0	0
fraggle	0	0	0	**smurf**	0	0	0
jolt	332	133	107	**teardrop1**	3	1	1
land	1	1	1	**teardrop2**	3	1	1
mydoom	0	0	2	**welchia**	2	5	5
nestea	6	2	2	**winnuke**	0	10	10
nimda	1	1	1				

Baselining Malicious Traffic. Tables 10 and 11 provide summaries of the alerts generated by Bro, Snort and Bleeding Snort for a single instance of each of the 21 MACE and 52 DARPA attacks. It is from the data used to create these tables that we produce the representation of the alarm types and frequencies we expect for each MACE and DARPA exploit. Certain exploits in these tables highlight some of the main differences between Bro and Snort, *i.e.*, Bro is generally concerned with stateful monitoring of connections and applications, while Snort is oriented toward detecting specific conditions in individual packets (such as the presence of a particular string). For example, a specific string in the **winnuke** exploit in Table 10 generates 10 alarms in Snort and Bleeding Snort but does not trigger any Bro alarms. Similarly, unexpected SMTP state pro-

[7]We verified these counts using a packet trace taken on a separate, unloaded host.
[8]This is exactly what is handled by payload-sanitize.

duced during the **mailbomb** exploit in Table 11 triggers 450 alarms in Bro, but 0 in Snort and 60 in Bleeding Snort.

Evaluating effect of Traffic Mix. Tests using a range of benign and malicious traffic mixes demonstrate a remarkable diversity in the behavior of Bro, Snort, and Bleeding Snort. Figures 5 and 6 show CPU utilization and packet loss results for the CSL/MACE and DARPA/attack-replay data sets. We see that as the overall flow composition becomes dominated by malicious flows, CPU utilization shows a clear increasing trend. We also see a difference caused by the application mix of benign traffic flows. For the CSL data set with no malicious flows (mix100), we see that Bleeding Snort consumes more CPU time than Bro, but that the opposite occurs for the DARPA data set with no malicious flows.

Packet loss behavior between the CSL and DARPA traffic is quite different for both Bro and Snort/Bleeding Snort. For both Snort variants, there is always some occurrence of packet loss. (We are currently investigating the cause of this high level of packet loss even with relatively low CPU utilization, which is consistent with previously reported experiments [21].) On the other hand, Bro appears quite resilient to dropping packets until it consumes all CPU resources, at which point loss becomes endemic.

Figures 7 and 8 show alarm effectiveness and efficiency over the range of benign and malicious flow mixes. We see that for the CSL data set alarm effectiveness drops as the traffic mix becomes dominated by malicious flows (false negatives increase). This effect is mainly caused by packet drops experienced by Bro and both Snort variants. False positives are relatively low across all mixes for the CSL data set, with the fewest false positives coming when most of the traffic is malicious (mix10/90).

For Bro with the DARPA data set, there is a low absolute number of expected alarms, but with the Snort variants, there are many more expected alarms. This low absolute number for Bro causes the large variation in false positives. Since the DARPA data set has existed for quite some time, it is possible that Snort has been tuned to perform well on this set of exploits. Note that we do not show any results for the mix100 (100% benign traffic flows) scenario for alarm efficiency or effectiveness since all alarms are considered false alarms.

Evaluating Effect of Volume. To understand the effect of volume in NIDS performance, we conducted experiments with traffic rates of 20, 40 and 60 Mbps. In these experiments, the mix of malicious to benign traffic was fixed at 10/90. Figures 5 and 6 show the packet drops and CPU utilization for the three systems under different volumes. On all systems, an increase in traffic volume directly affects CPU utilization. Interestingly, while Bro seems quite resilient to packet drops with the CSL data set, the DARPA data set, which is dominated by HTTP traffic, has a substantial impact on Bro performance. Snort's drop rates seem to degrade less intensely with volume for this data set.

In Figures 7 and 8 we show the effectiveness and efficiency of the systems in the volume experiments. Efficiency and effectiveness do not seem to be highly correlated with volume of traffic although counts of alarms do increase with volume. Second, it seems that Snort's signature set has been to tuned to detect DARPA attacks much better than Bro. Bro generates few alarms on the DARPA data, and as a result even though Bro produces few false alarms its efficiency and effectiveness are poor. For the CSL traffic, Bro's efficiency and effectiveness are significantly better. Another observation is that Bleeding Snort rule set, which is a superset of the Snort rule set, seems to have lower efficiency and effectiveness. As we would expect, the volume of baseline alerts in Bleeding Snort is higher than for Snort. As a result, Bleeding Snort has a greater chance of missing true alarms during packet drops but also a larger chance for false positives, which decreases effectiveness.

Discussion. We believe that our results demonstrate that Trident is well-suited to test NIDS that consider protocols and that it provides a unique set of capabilities for testing the class of networked systems that maintain connection state. The results also suggest a range of possibilities for tuning and configuring NIDS at a local site based on expected average or worst case traffic scenarios. Even though the overall traffic volume of any experiment is not very high, each NIDS configuration exhibits a wide range of performance characteristics over the set of tests. This diversity in performance reveals the importance of the mix of benign application traffic and malicious traffic. Our experiments also show that our decision to interleave payloads that originally belonged to different flows (while respecting a given service automaton) has no discernible impact on alarm quality. Furthermore, the ability, via packet interleaving, to create flows of virtually any size and mix facilitates exercising NIDS over a broad spectrum of conditions.

With Trident, the system resource requirements of an IDS configuration can be tested in an emulated live network setting. Typically, offline evaluations using publicly available data sets or locally captured packet traces can reveal only limited aspects of total system performance. A key consequence of the capabilities provided with Trident is that comprehensive system evaluation can be done using locally captured packet traces enabling tests over a range of relevant site-specific conditions.

Lastly, our results expose a key challenge in designing NIDS: graceful degradation under unexpected or extreme conditions. For example, comparing results of the CSL and DARPA data sets shows that the the overall mix of application traffic has a significant impact on NIDS system performance, and that this mix can also greatly affect alarm quality. Even with relatively low overall traffic volumes, Trident can easily push each IDS to its operating threshold.

8. CONCLUSION

In this paper, we describe a traffic generation framework for robust online evaluation of network intrusion detection systems. The objective of our work is to create a system that generates realistic, diverse streams of both benign and malicious traffic. We develop a methodology for benign traffic generation based on service specific automata and using packet payloads from either empirical packet traces or from the standard DARPA traces. This approach is likely to preclude exact reproduction of experimental results since privacy concerns will limit sharing across sites. However, we argue that it enables highly representative testing and will be highly useful for researchers developing new systems and security administrators seeking to test and tune their own systems. We implemented our traffic generation framework in a tool set we call Trident. We demonstrate the utility of Trident through a set of experiments on open-source NIDS conducted in a controlled laboratory setting. The results of these experiments indicate that content, mix and volume have tremendous effect on NIDS performance and demonstrate Trident's capability to expose a range of diverse behavior on modern NIDS.

9. REFERENCES

[1] Flowreplay Design Notes. http://www.synfin.net, 2003.
[2] BleedingSnort. http://www.bleedingsnort.com, 2005.
[3] Metasploit project. http://www.metasploit.com, 2005.
[4] Skaion's Traffic Generation System (TGS). http://www.skaion.com/products/index.html, 2005.
[5] THOR: A Tool to Test Intrusion Detection Systems by Variations of Attacks. http://thor.cryptojail.net/, 2005.
[6] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Toulouse, France, October 2000.

Table 11: Counts of alarms generated by Bro, Snort, and Bleeding Snort for a single instance of each DARPA exploit.

exploit	Bro	Snort	Bleeding Snort	exploit	Bro	Snort	Bleeding Snort	exploit	Bro	Snort	Bleeding Snort
apache2_one	2	0	0	**named**	0	2	4	**sendmail**	7	0	2
casesen	2	8	10	**ncftp**	3	6	8	**smurf**	0	0	0
crashiis	1	2	4	**neptune**	12	2	10	**snmpget**	0	1505	1505
dict	0	4	4	**netbus**	5	0	6	**sqlattack**	0	0	1
dosnuke	0	7	7	**netcat**	5	0	6	**sshprocesstable**	0	0	100
fdformat	1	0	0	**ntinfoscan**	7	28	38	**sshtrojan**	0	0	0
ftpwrite	5	2	4	**perl**	1	0	2	**syslogd**	0	0	0
guessftp	0	0	5	**phf**	3	3	3	**teardrop**	0	6	4
guesspop	0	0	0	**pod**	0	12	12	**udpstorm**	0	0	0
guesstelnet	0	80	120	**portsweep**	0	5	5	**warezclient**	1	0	2
guest	9	34	50	**processtable**	0	0	54	**warezmaster**	1	0	2
httptunnel	0	1	1	**ps**	1	9	7	**xlock**	0	4	4
illegalsniffer	0	0	13	**queso**	0	2	2	**xsnoop**	0	2	2
imap	0	2	6	**resetscan**	0	0	0	**xterm**	1	0	2
land	0	2	1	**satan**	4	4	4	**xterm1**	1	0	2
loadmodule	0	0	0	**sechole**	2	23	26	**yaga**	4	6	8
ls	0	2	2	**secret**	0	0	1				
mailbomb	450	0	60	**selfping**	1	2	2				

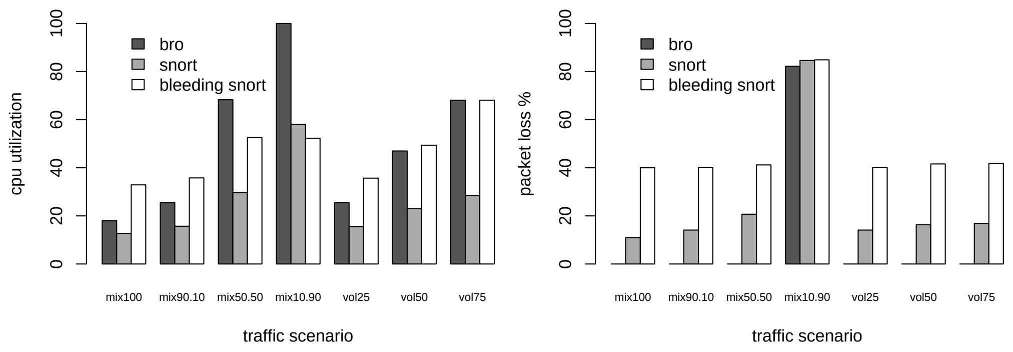


Figure 5: CPU utilization (left) and packet loss (right) measurements for Bro, Snort, and Bleeding Snort on CSL traffic setup.

[7] S. Antonatos, K. Anagnostakis, and E. Markatos. Generating Realistic Workloads for Network Intrusion Detection Systems. In *Proceedings of ACM Workshop on Software and Performance*, Redwood Shores, CA, January 2004.

[8] D.Mutz, G. Vigna, and R. Kemmerer. An Experience Developing and IDS Simulator for Black-box Testing of Network Intrusion Detection Systems. In *In Proceedings of ACSAC*, Las Vegas, NV, December 2003.

[9] Exploitation Framework. http://www.securityforest.com/wiki/index.php/Exploitation_Framework, 2005.

[10] S.-S. Hong and S. F. Wu. On Interactive Traffic Replay. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Seattle, WA, September 2005.

[11] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

[12] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating Intrusion Detection systems: 1998 DARPA Off-line Intrusion Detection Evaluation. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 1998.

[13] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 DARPA Off-Line Intrusion Detection Evaluation. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Toulouse, France, October 2000.

[14] M. Mahoney and P. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Intrusion Detection. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Pittsburgh, PA, September 2003.

[15] J. McHugh. Testing Intrusion Detection Systems: A critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4), November 2000.

[16] Nessus. http://www.nessus.org, 2005.

[17] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proceedings of ACM SIGCOMM/Usenix Internet Measurement Conference*, Taormina, Italy, October 2004.

[18] D. Plonka. Flawed Routers Flood University of Wisconsin Internet Time Server. http://www.cs.wisc.edu/˜plonka/netgear-sntp, 2003.

[19] L. Rossey, R. Cunningham, D. Fried, J. Rabek, R. Lippman, J. Haines, and M. Zissman. LARIAT: Lincoln Adaptable Real-Time Information Assurance Testbed. In *Proceedings of IEEE Aerospace Conference*, Big Sky, Montana, March 2002.

[20] J. Sommers and P. Barford. Self-Configuring Network Traffic Generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, Taormina, Italy, October 2004.

[21] J. Sommers, V. Yegneswaran, and P. Barford. A Framework for Malicious Workload Generation. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference*, Taormina, Italy, October 2004.

[22] S. Staniford, J. Hoagland, and J. McAlerney. Practical Automated Detection of Stealthy Portscans. *Journal of Computer Security*, 10(1-2), 2002.

[23] K. Wang and S. Stolfo. Anomalous Payload-based Network Intrusion Detection. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Sophia Antipolis, France, September 2004.

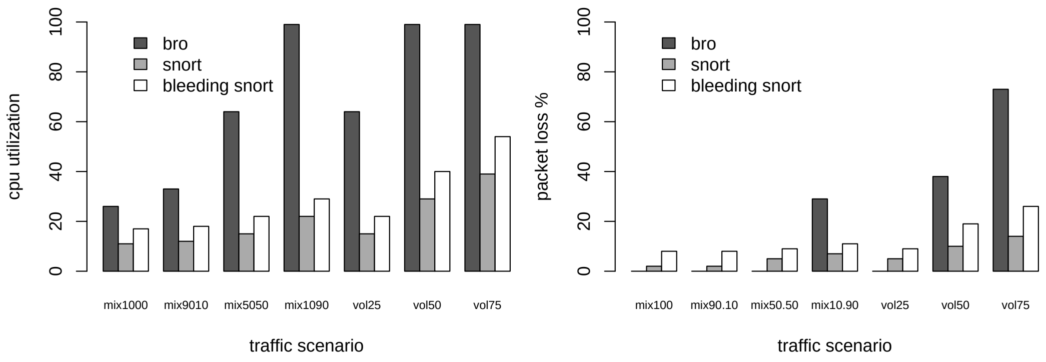


Figure 6: CPU utilization (left) and packet loss (right) measurements for Bro, Snort, and Bleeding Snort on DARPA traffic setup.

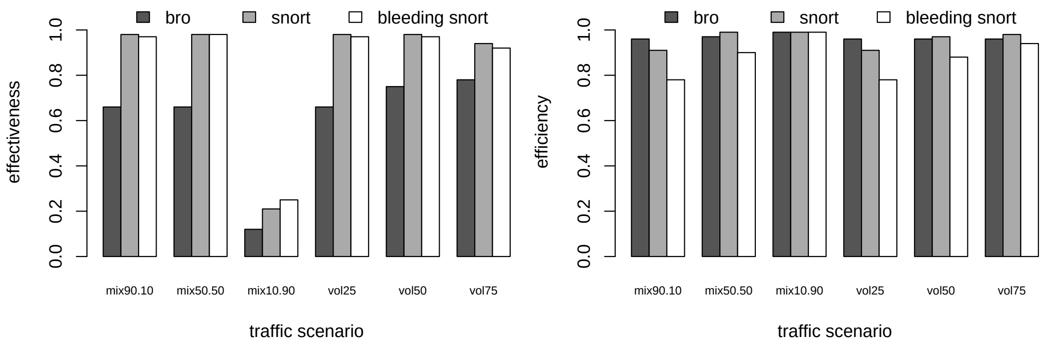


Figure 7: Alarm effectiveness (left) and efficiency (right) for Bro, Snort, and Bleeding Snort on CSL traffic setup.

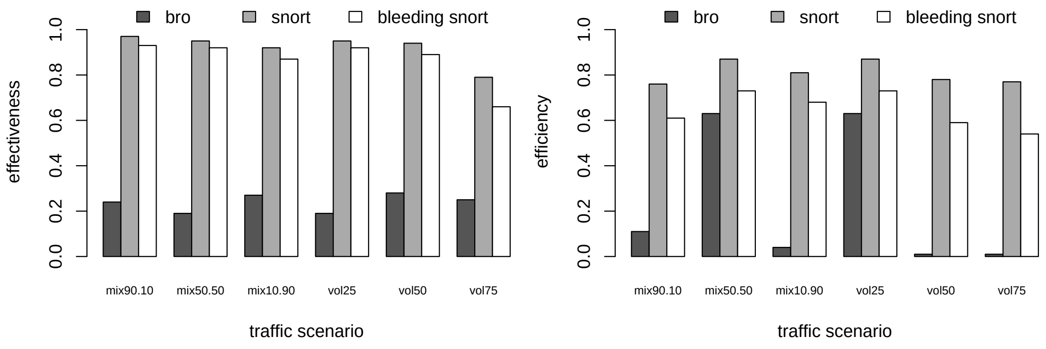


Figure 8: Alarm effectiveness (left) and efficiency (right) for Bro, Snort, and Bleeding Snort on DARPA traffic setup.