Intrusion Detection Testing and Benchmarking Methodologies

Nicholas Athanasiades, Randal Abler, John Levine, Henry Owen, and George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0250 USA

Abstract— **The ad-hoc methodology that is prevalent in today's testing and evaluation of network intrusion detection algorithms and systems makes it difficult to compare different algorithms and approaches. After conducting a survey of the literature on the methods and techniques being used, it can be seen that a new approach that incorporates an open source testing methodology and environment would benefit the information assurance community. After summarizing the literature and presenting several example test and evaluation environments that have been used in the past, we propose a new open source evaluation environment and methodology for use by researchers and developers of new intrusion detection and denial of service detection and prevention algorithms and methodologies.**

***Index Terms*—Intrusion Detection Testing, Network Security, Network Attack Generation, Hacker Tools**

I. Introduction

Beginning in 1998, DARPA initiated the Intrusion Detection Evaluation [8-12] program. These evaluations involved generating background traffic interlaced with malicious activity so that intrusion detection systems and algorithms could be tested and compared. Although these tests and the resulting data were very valuable, after 1999 this evaluation environment and methodology has not been continued. In 2000, the Lincoln Adaptable Real-time Information Assurance Test-bed (LARIAT) effort was initiated. This effort resulted in custom software that emulates network traffic from a small network connected to the Internet. By 2001, LARIAT had the ability to perform approximately 50 attacks against 9 operating systems [26]. Unfortunately, LARIAT is only available for use under special circumstances, thus it is not available to the Information Assurance community as a whole.

Since neither LARIAT nor an updated DARPA environment is openly available, inventors of new information assurance algorithms and techniques have no accepted, standard method for evaluating their algorithms and implementations. The most common methodology is to create a small test network and release malicious activity in that self-contained small network environment. The network architecture, complexity and capabilities are totally unspecified in general, and no standard comprehensive recommendations appear to exist. A second methodology also used is to capture network traffic from an existing production network and play it back on a test network segment using tools such as "*tcpreplay*" [47]. The disadvantage of this second approach is that new attacks are not easily inserted into the existing traffic data files. Neither of these approaches has been widely effective in evaluating intrusion detection or intrusion prevention systems, due to the inability to generate network traffic data from a large variety of environments and conditions.

The use of "complete traffic" is necessary for realistic information assurance system testing [22]. In general the issue of traffic generation is one of the most difficult ones to tackle. Synthetic traffic does not represent the realities of an actual network. Synthetic load generators, like SmartBits hardware traffic generators, were designed for load-testing bridges and routers, which typically do not examine packet payloads [22]. They produce pseudo-random TCP frames. The traffic generated by these tools is usually not suitable for use in information assurance evaluation environments, since actual traffic is not random. Statistical models that take into account daily patterns of computer usage with random deviations may be used to describe user behavior, and have been be used to extrapolate the traffic produced by legitimate users. However, the user models produced to date have been site and application specific and do not carry the generality necessary to be useful for evaluation purposes.

Software testing approaches have used scripting tools to create synthetic "Web browsers", as well as FTP and Telnet users. Their sessions last random periods of time. These methods seldom model actual user behavior, and thus this approach is nearly equivalent to the random packet generation approach. Traffic generators designed for testing web servers have more realistic profiles because the type of traffic created is by design like Internet traffic [22]. In testing web servers, the offered load becomes an issue because overloading a server resembles a DOS attack. More over, types of URL and data returned from the server influence the response of the traffic generation system. Scripting tools are commonly used to emulate users engaging in browsing the Internet.

Reviewing the intrusion detection system testing and benchmarking literature, it becomes clear that present approaches to testing are inadequate. A large number of unrelated tests, such as [29,3,32,34-39,43-45], can be brought into use. Most of these tests assess susceptibility to a single weakness or single weakness area. We do not believe that there is ever going to be a comprehensive test nor a single

©2003 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

methodology that would examine an intrusion detection system for every attack, verify its abilities and identify all its weaknesses [28]. This in essence would categorically solve the problem of intrusion detection. However, it is possible to improve on the existing practices employed in the field. In particular, a common framework for testing, as in this paper, would greatly simplify and expedite running a wide variety of test scenarios.

II. EXISTING TOOLS AND TESTING METHODOLOGIES

The predominant open source philosophy for testing intrusion detection systems in 2002 is to create an individualized test environment. This means searching for existing exploits (attacks) and then embedding these exploits into the individualized test networks. Background traffic is mixed with the exploits to hide the exploits in benign environments. Volume and traffic data rates are controlled so as to have the ability to determine at which points the intrusion detection algorithm or platform begins to have trouble keeping up the total traffic. While not presenting an exhaustive list of these environments or individual tools, the next subsections summarize a few of the tools and environments used for creating intrusion detection evaluation environments. A more comprehensive list of tools may be found at [49] which is a list of the 50 most popular network security tools.

A. DARPA Environment

The DARPA 1998 and 1999 intrusion detection evaluations represent the first significantly systematic effort to test intrusion detection systems.Their focus was not on testing complete systems but evaluating technical approaches [26]. In the first evaluations in 1998, the state of intrusion detection was unknown. The designers of the environment had to overcome the considerable problem of no pre-existing effort. No previous effort meant that there were no standard comparison metrics, standard attacks, standard background traffic or existing methodology [10].

The general approach to testing was the following. An off-line and an on-line evaluation were executed. In the off-line evaluation, a set of seven weeks of training data was provided to vendors. This consisted of sessions, which were marked as normal, or as attacks. The off line training data was intended to use as a tool for the intrusion detection system experts to tune and optimize their systems [11]. A second set of two weeks of data was generated which was to be used for actual testing. The tool *tcpreplay* was used to feed the traffic to the evaluation systems. The results were analyzed and presented to participants at a workshop [10].

The test designers were faced with three options when deciding how to generate the traffic. The first option was to collect real operational data and attack an actual organization. The packets would be real, however it was unacceptable to attack an actual organization. Furthermore, releasing private e-mails, passwords and user identities was not realistic. The next option was to actually go ahead and sanitize the data collected and then introduce the attacks directly in the sanitized data [5]. This approach was cumbersome and prone to artifacts since attacks would be merged into real collected traffic. There was no other realistic option other than to synthesize all sessions from scratch. This option would generate non-sensitive traffic similar to that seen on an operational network. Data could be distributed without fear of breaching security. More over, it lent itself to automating traffic generation. In order to recreate realistic traffic four months of data from Hanscom Air Force Base and 50 other bases were analyzed [10].

The network traffic in this environment was Unix focused. It included the following protocol/traffic activity: HTTP, X Windows, SQL, SMTP, DNS, FTP, POP3, Finger, Telnet, IRC, SNMP, and Time [10]. Thousands of hosts contributing to background traffic were simulated but the number of simulated users representing secretaries, programmers, workers and so on were in the hundreds. In the cases where traffic was too complex to model with automata, human actors were utilized to generate background traffic. The actual content of FTP, HTTP and SMTP transfers was created through statistical similarity to live traffic or was sampled from public–domain sources [5]. A number of e-mails in the network traffic were taken from public domain mail list servers. The rest maintained word and two-word sequence statistics derived from a sample of 10,000 e-mails filtered with a 40,000 word dictionary to remove names. Telnet sessions were recreated via statistical profiles of users. These profiles included the frequency of UNIX commands executed, typical login times, session duration times, source and destination addresses. Some of the computers maintained fixed IP addresses while others changed addresses during the test [5]. Expect was used to control most of the automated sessions. Human actors performed more complicated tasks such as installing software. In order to complete the testing environment, attacks needed to be interjected into a volume of background traffic. 120 different attacks spanning 38 attack categories were introduced into the traffic [10]. Figure 1 presents the attacks launched against each victim.

	Solaris	SunOs	Linux
Denial of Service (11 types, 43 instances	Back,Neptune, Ping of death, Smurf, syslog, Land, apache2, Mailbomb, Process table, UDP storm	Back,Neptune, Ping of death, Smurf, Land, apache2, Mailbomb, Process table, UDP storm	Back,Neptune, Ping of death, Smurf, teardrop, Land, apache2, Mailbomb, Process table, UDP storm
Remote to Local (14 Types, 17 Instances)	Dictionary, ftp-write, guest, phf, http tunnel, xlock,xsnoop	Dictionary, ftp-write, guest, phf, http tunnel, xlock,xsnoop	Dictionary, ftp-write, guest, imap, phf, named, http tunnel, sendmail, xlock,xsnoop
User to Root (7 types, 38 Instances)	Eject, ffbconfig, Fdformat,ps	Loadmodule, ps	Perl,xterm
Surveillance/ Probe (6 types, 22 Instances)	Eject, nmap, Port sweep, Satan, mscan, saint	Eject, nmap, Port sweep, Satan, mscan, saint	Eject, nmap, Port sweep, Satan, mscan, saint

Figure 1 Attacks in the 1998 DARPA evaluation [10]

In addition to the methodology of using known attacks, novel attacks were developed in order to test systems' effectiveness against never-seen before attacks [10]. Many of the attacks were analyzed beforehand to verify that they function in the test bed. That information was later used further to improve on these attacks by making them stealthier [5].

The suggestions collected during the 1998 test shaped the 1999 DARPA test. The goals of the evaluation shifted to testing complete systems and the reasons they miss novel attacks. There were a number of changes and additions summarized below [8]:

1. Victim Windows NT machines were added.
2. New stealthy attacks were added to avoid intrusion detection system detection.
3. Two new types of analysis were performed:
   a. An analysis of misses and high-scoring false alarms to determine the cause of detection misses and false alarms.
   b. Participants were allowed to submit information aiding in the identification of many attacks and their appropriate response.
4. Another major point of focus was detection of novel attacks without first training.

Scoring was restructured for the 1999 test. The list files as generated for the 1998 test made it difficult for the vendors to submit easily scored alerts because of the complex nature of alert generation. Alerts were true positives if they occurred at the time of attack and the correct victim's IP was identified. A 60 second grace period was allowed to account for network latencies. Intrusion detection systems were not penalized for attacks for which they were not designed to detect [8].

The DARPA tests have been criticized extensively. The environment was build to test passive intrusion detection systems. The off-line part was very difficult to adapt for systems that query network equipment and respond by changing their configuration. Furthermore, the information recorded for the off-line test was simply not enough for certain intrusion detection systems. In general, recording packet streams have proven inflexible as a testing method. The complexity of packet streams necessary to test correlation systems makes the DARPA approach not extendable [26].

The DARPA data generation methodologies have been critiqued because one of the expressed goals of the DARPA evaluation was to create useable data for further testing by others. Criticisms included [15]:

1. No effort was made to validate its false alarm characteristics. It is unknown how many false alarms background traffic alone would generate.
2. Neither the statistics of the real traffic nor the statistics that the generated traffic was supposed to match, or the methodology for proving the statistics correct, were ever published.
3. Data rates and their variation with time was never a variable in the DARPA tests. What is more troublesome is that there is suspicion that data rates in the order of kilobits may have been used.
4. The attacks were evenly distributed throughout the background traffic. Each attack type was used the same amount. Both choices do not reflect network reality.
5. There are inconsistencies in the documentation found on the DARPA tests.
6. There was no control group employed in the testing.
7. The types of attacks used could guide one's detection focus on operating systems vulnerable to interactive attacks.
8. The size of training data may have been insufficient. There is no proof that it presented intrusions to any degree of realism or even that such data can be constructed.
9. Some attacks had TTL characteristics that did not match the background traffic.

In spite of these criticisms, the DARPA tests and methodology were a significant contribution to the information assurance community and made significant contributions toward identifying the complexities and difficulties in testing and evaluating intrusion detection algorithms. Test data is still used at present in 2002. However, at present, the information assurance community frowns upon further results and comparisons made with this data. This is due largely to the fact that the time value of the actual data has depreciated both through new and different attacks today as well as having more time to customize and calibrate intrusion detection algorithms. This means that it is not fair to compare to results from algorithms used during the time pressure of the DARPA tests. It is undesirable that there is no presently publicly available existing equivalent environment or methodology available for researchers to use to evaluate the new generation of algorithms and methodologies. These DARPA tests were a very positive influence on the information assurance community.

B. LARIAT Environment

A recently developed follow on methodology and environment is the Lincoln Adaptive Real-time Information Assurance Test-bed (LARIAT) [23]. The motivation for creating LARIAT was to provide "tools to assist in the evaluation and configuration of information assurance (IA) technologies" [23]. This was part of a larger effort to develop a comprehensive testing environment for intrusion detection systems, firewalls and access control lists. It came about after the DARPA 1999 IDS evaluation with the first release becoming available in 2000. In 2001, its capabilities had been extended to include high throughput capabilities, attack scenarios and Windows traffic in addition to being real-time, deployable and fully automated.

LARIAT "emulates the network traffic from a small organization connected to the Internet" [23]. The user first selects profiles for background traffic and attacks. The attack profiles include the type and strike time of the attack. The

system goes through a network discovery phase during which it verifies that everything is ready before the test begins. In the future, test engineers will not have to configure LARIAT in order to specify the capabilities of the network. The system will do that by itself [23]. Then, it initializes the network (clears logs and process table, removes old traffic and resets accounts, etc.) and configures the hosts. During the next step the test's conditions are set up. Traffic and attack scripts are generated. Attacks are scheduled and logging services are started. The software allows the configuration of the aggregate traffic generation including the attack distribution, rate and amount of traffic. Subsequently the test is run. LARIAT allows the user to monitor the testing progress in "real-time". After completion of the test the attack logs are examined and their success evaluated. The final stage consists of clean up. All corrupted files and polluted process tables are reinstated. Services and other conditions necessary for the test's execution are terminated [23]. The system is fully automated so after clean up the system will start the next attack scenario by repeating the process from step two.

Traffic generation in LARIAT is done through the use of defined service models. Some examples of the protocols used are http, smtp, ftp, telnet and much more. Further more the system takes into account interactions between protocols. Traffic volumes can be varied from 0-100Mbps according to need. In 2001, there were 50 attacks available for 9 different operating systems.

The system takes a unique approach to background traffic generation. The developers have modified a Linux Kernel that would allow their software to generate traffic. The traffic emulates many hosts on the Internet. There is a web interface that allows evaluators to configure the traffic generation between individual tests [26]. The traffic used is similar to that used by the DARPA 1999 evaluations. The user is allowed to adjust the arrival rate and distribution of sessions of each type [26]. More over, there are traffic profiles, on/off time switches and zones can be added or removed. Zones are Internet domains or sub domains which contain real or virtual hosts, sources and destinations. All this information is supplied to traffic generators in order to adjust their traffic profiles. In conjunction with the ability of LARIAT to set a victim machine to a particular state (certain users, accounts and directories), the aggregate content of background traffic can be completely specified permitting the emulation of a wide variety of environments.

LARIAT was generated as part of a government project and is not publicly available. LARIAT is a very sophisticated and advanced test and evaluation environment and serves as an excellent reference for ideas that need to be incorporated into an open source tool that is widely available to all information assurance researchers and developers.

C. Nidsbench and IDS Wakeup

Nidsbench is a Network Intrusion Detection System Test Suite that has not been significantly changed or updated since it was first released in 1999. It is a tool kit that was intended for use in testing network intrusion systems and algorithms. It assumes that the systems under test are passive. In other words, the intrusion detection systems do not respond to the network traffic by adjusting the configuration of system [38]. Nidsbench is made up of the components *tcpreplay*, *idtest* and *fragrouter*. *Fragrouter* was used to detection if an attacker could evade the IDS as it is described in [30]. *Tcpreplay's* purpose was to provide the background traffic by replaying prerecorded traffic. *Idtest* is the strength of this tool in that it actually attempts exploits as opposed to vulnerability scanners that look for only for attack symptoms [38].

D. IDSwakeup

IDSwakeup is another group of tools built to test intrusion detection systems [43] very much like Nidsbench. It generates false attacks, which resemble well known attacks in order to determine if the systems will produce false alarms. It should be clear that this is not a scanner but a false positive test utility [44]. It is published under a BSD-style license [43]. The program consists of IDSwakeup and utilizes *hping* and *iwu*. IDSwakeup is the starting script and allows the user to choose which attack or attacks to imitate. *Hping* is used to send arbitrary packets. *Iwu* sends a buffer as a datagram. Source and destination addresses and TTL are changeable.

E. Flame Thrower

Flame Thrower is commercial load stress tool used to identify network infrastructure weaknesses. It produces actual transactions in order to test network infrastructure and applications. It supports HTTP/HTTPS 1.0, 1.1 and SSL with over 32,000 URL definitions. It can emulate over two million IP addresses [25]. FirewallStressor, which is part of the Flame Thrower environment, will measure throughput under attack conditions such as SYN Flood Attack, IP Spoofing Attack, CRC Error Packets, ICMP attack, LAND attack, Ping of Death attack, Trin00, Short Frame Packets, Tear Drop Attack, Illegal TCP/IP Packets, and Long Frame Packets. Flame Thrower is intended for testing firewalls [25].

F. WebAvalanche/WebReflector

WebAvelanch and WebReflector are two commercial network appliances, which are used in the testing of intrusion detection systems [24]. The former is a stress-testing appliance while the latter emulates the behavior of large Web, application and data server environments [39].

WebAvelanche and WebReflector support protocols such as HTTP 1.0/1.1, SSL, RTSP/RTP and FTP while allowing for the modeling of user behavior. The environment can maintain over a million connections, which will appear to come from different IP addresses. The environment will measure percent dropped packets, latencies, maximum number of users and new user arrival rates [39].

G. Tcpreplay

Tcpreplay is a utility that allows captured traffic to be

played back on a network at different speeds. According to [47] "this program was written in the hopes that a more precise testing methodology might be applied to the area of network intrusion detection, which is still a black art at best". Tcpreplay replays files captured in tcpdump or snoop formats.

H. Fragrouter

This is an attack generation tool. It is recommended for testing anti-evasion techniques and fragmentation queues [24]. "Fragrouter is a program for routing network traffic in such a way as to elude most network intrusion detection systems" [24]. In [24] techniques using fragmentation to elude intrusion detection systems were explained.

I. Hping2

Hping2 is a command-line packet assembler and analyzer [29]. Hping2 is distributed under GNU GPL license for Linux, FreeBSD, NetBSD, OpenBSD and Solaris platforms. Hping2 allows one to create and transmit custom ICMP, UDP, and TCP packets. Hping2 may be used to fingerprint remote operating systems.

J. Iperf

Iperf measures bandwidth, delay jitter and datagram loss. Measurements can be done using representative traffic streams. Iperf may be used as a background traffic source for intrusion detection environments [33,34].

III. Issues in Generating Realistic Evaluation Environments

The problem of traffic generation may be divided is into two issues. The first is concerned with background traffic. It consists of flows that do not carry malicious payload. The second issue tackles the actual testing of intrusion detection systems with attacks. These two types of flows are mixed and presented to an intrusion detection system for processing. The decoupling of the two types of flows has allowed intrusion detection evaluators a great degree of flexibility. It is generally accepted that the most optimal background traffic is unprocessed, real flows encountered in an actual network. It is abundant. It requires no effort from the tester to generate. It could not be more realistic. It contains no harmful artifacts such as duplicate addresses coming from different locations, which would confuse some switches. It is easy to vary.

The attack traffic, on the other hand, is based on databases, which serve as repositories for malicious flows [6,7][46]. These databases require effort to maintain and update. Attack density can be changed real time, which is impossible with canned traffic. There is real time generation. The traffic is easy to vary. The insertion of "cessation times" is relatively simple. Cessation times between attacks are relatively important in order to distinguish between the effects of attacks when those are similar. Again evaluators have full control of the attack stream.

At the moment, it is very expensive to run an exhaustive test of all attacks known. There are diminishing returns in the confidence of an intrusion detection system after a certain number of attacks have been attempted. A suggestion is to attempt a representative subset of attacks from each category. This is called equivalence partitioning [19]. However, that introduces the problem of taxonomy [40]. Current work seems to indicate that classification methods using vulnerabilities, signatures and intrusion techniques to lead to imperfect class definitions [19]. Both of the schemes would ensure that a wide range of test cases would be selected. Therefore, it is recommended to use the security policies already in place as guidelines in the determining which attacks to launch again the intrusion detection system under evaluation. Existing taxonomies should be employed as a tool to guarantee the variety of the attacks [18].

Another disadvantage has to do with regulatory restriction on user traffic. The reason DARPA created "pseudo users" was that actual traffic contained sensitive or classified information. Privacy laws may require sanitizing traffic before use for testing, thereby invalidating it's utility.

The authors of this paper requested permission from our own academic organization to sniff and analyze the traffic on our own subnet going into and out of our own research laboratory network that consists of approximately 20 machines behind our own Ethernet switch. This request was denied. Academic departmental computer support personnel gave the reason as that of data "privacy". Thus, we were unable to gain access to our own machine's traffic for the purposes of network intrusion algorithm development.

There are a number of environmental factors which affect evaluation results and which should be isolated as part of the testing methodology. The effect of networking applications such as firewalls, proxy servers and shared networks restrict the types of traffic encountered in the network. A system whose attributes are to be evaluated should be tested in different network environments. For example in some network environments, non-symmetrical routing may be in use. In this type of network environment, only inbound or outbound traffic may be seen. This will radically affect a network intrusion detection algorithm's capability. In any event the network architecture and topology used in the evaluation must be selected carefully. Different algorithms and methodologies will perform differently in different network topologies [13].

An environment related suggestion is to avoid using hosts which carry the intrusion detection systems under test as victims. If the use of the host running the intrusion detection system as a victim is necessary, multiple tests varying the underlying operating system are suggested. It is conceivable that the performance would vary as well. However, the proposition is based on the fact that current operating systems lack basic security mechanisms (trusted path and protected

path [28]), which imply that security software level on the application is built on "sand" [28]. Attacks could cause the intrusion detection systems crash and thus incomplete testing. Furthermore, attacks could change the environment the intrusion detection system operates in by effecting other network equipment such as router and firewall [28] [13].

Currently testing is limited to case-by-case scenarios. Components and specific situations can be evaluated but true-life performance conclusions would not be advised. The use of real network data as background to injected attacks provides closer to reality response and is highly recommended as an alternative to pseudo traffic generators. The 1998 and 1999 DARPA approach was a significant effort to intrusion detection benchmarking unmatched in its attempt to provide a realistic environment for evaluation. However, it has inadequacies and is not a good approach by itself. The traffic traces generated from that effort and later published required a lot of rework in order to be reused by others. In addition, the statistical models for the users that created the background traffic were approximations that do not generalize well.

In an effort to side step the high cost of creating new user models every time a new test is needed, tests attempted to reuse the traffic traces from previous tests. More specifically, an attacking machine replays the traffic (using a tool such as tcpreplay) recorded during a live test. A number of sites on the Internet published traces of attack traffic from "hackathons" and other tests. One of those sites is [42]. According to [8] more than 90 sites downloaded all or part of the 1998 packet traces from the Lincoln Labs website with the intension to use it as aid to new product development. Speeds for traffic generation are limited by hard drive access times. High-speed problems occur beyond the 100 MB traffic because of the bandwidth of hard drives. Problems start at 20-40 MB/sec [22]. It is not possible to conduct high speed, high volume tests with these tools. Additionally, it is hard to use this type of traffic when the network contains components that respond to changing conditions on the network by adjusting the configuration of different equipment. Changes in configuration would have no effect given that the traffic stream is fixed [11][26].

IV. Examples of Intrusion Detection Evaluation Environments

In the following subsections, we summarize five environments that have been used to evaluate intrusion detection systems. While these are not all of the example environments that we are aware of, they are representative of the techniques being used.

A. *DARPA Like Environment*

In the first example, a comparison of a signature based system and a DARPA research system relying on statistical techniques was desired. The environment focused on DOS attacks. The evaluation environment was trying to answer the following questions:

- Are there regions of operation where the attack tools can degrade performance while escaping detection? [3]
- Does the intrusion detection system with the statistical techniques have an advantage over the signature base system?
- In the case that the intrusion detection system had the ability to respond would it be effective?

The dependent variables used were:

- Sufficiency of information provided by the system for diagnostic purposes.
- Proper identification of attack source.

The test set up consisted of 5 components:

- Traffic was generated in the same way as in the DARPA 1998 test bed [4].
- The victim machine was "an anonymous FTP server running on a Sun UltraSparc-1 using a Solaris 2.5 operating system." [3]
- Attack Injection programs placed attacks on the network in a scalable and predictable manner degrading the performance of the victim machine proportionally to the level of the attack. An in house tool was used which opened up connections on the server without closing them. The rate of attacks was adjustable in order to test of minimum level of detection [3].
- The in house reference programs counted the number of hung connection at the victim server as a measure of attack effectiveness. They used a metric called virulence. Virulence described the intensity of an attack situation [3].
- The evaluation method was to use 10, 15, 30, 40 and 60 attacking hosts each utilizing rates of varying rates of attacks per second. The two systems were evaluated under 20 test conditions. In permutations the percentage of detected hosts was measured [3].

B. *Custom Software*

In the second example, a software platform was developed that simulates intrusions and tests IDS effectiveness [18]. This approach to the problem was to evaluate an intrusion detection system just like any another application. The evaluation criteria that were used included [18]:

- Broad Detection Range: measured the ability of the system to detect different type of intrusions.
- Economy in resource usage: measured consumption of computer resources by the intrusion detection system.
- Resilience to stress: looked at operational impairment in the case of high computing activity.

The benchmark platform was base on Expect and Tool Command Language Distributed Programming (TCL-DP) package. Expect was used to simulate users performing basic operations such as Telnet and FTP. In addition, a record-and-replay feature supplemented the creation of tests through the replay of script execution. Close attention was placed on concurrency and control in order to produce repeatable tests and results [18]. The intrusion detection system was installed on a Sun workstation that was connected to a LAN segment. The testers performed attack identification tests during stress tests. Some of the attacks involved password file transmission to remote hosts, password cracking, and password dictionary attacks and exploits go gain super user access [18].

C. Advanced Security Audit Trail Analysis on Unix

The third example was part of an experiment concerning the evaluation of distributed intrusion detection systems called Advanced Security audit trail Analysis on uniX (ASAX). The purpose of the evaluation was to assess the reliability and efficiency of ASAX [1]. The test is interesting for two reasons. The test network was part of an actual network. The tests took place at a student cluster of Unix computers. Secondly, ASAX performed only the analysis of the data not the actual collection. The test consisted of the following scenarios [1]:

- Trojan horse: executable files with commands accessing unauthorized paths.
- Attempted break-ins: unsuccessful connection attempts.
- Masquerading: many identity changes.
- Suspicious network connections: connections received from
- Black listed addresses.
- Nosing: numerous moves through directories.
- Privilege abuse: remote connections reading files such as netrc.
- Exploitation of an lpr flaw.
- Leakage: consultation of certain files.

Measurements concentrated on a central machine. Performance was determined with and without the intrusion detection system for the duration of a month. The detection ability of the system was evaluated based on the intrusions the testers tried.

D. Vendor Independent Testing Lab

The fourth example is an environment created by the NSS group, an independent network and security testing facility. They perform Intrusion Detection Systems testing and have issued a test report, which contain information on 16 IDS systems [41]. NSS tests a broad range of features of intrusion detection systems. They look at convenience, which include ease of installation, deployment and management. Moreover, they are interested in the user interface; how is the reporting occurring and alerts delivered. Part of the evaluation is the enforcement of the company's security policy through the IDS. Attack signatures are important for many systems, so a portion of their tests is dedicated to finding how many are supported, whether custom ones are allowed and how these are updated. Some systems provide corrective action during an attack. Thus NSS tests the effectiveness of the methodology responses. Emphasis is placed on a related issue such as forensics. The abilities of an intrusion detection system to capture, provide protocol and record an attack are appraised. Accuracy and depth of prevention advice is another metric. Finally, peripheral issues like licensing, documentation and log management are looked at [41].

The NSS's test set up consisted of Pentium III 1GHz PCs each with 768 MB RAM running Windows 2000 SP2, FreeBSD 4.4 or Red Hat 6.2/7.1 were used for the tests. Each machine was clean installed after each test from a Symantec Ghost image. The computers were connected via 100Mbit Ethernet with CAT 5 cabling, Intel NetStructure 40T routing Switches and Intel auto-sensing 10/100 network cards. Intel provided all the necessary drivers. Each intrusion detection system was installed on a dual-homed PC on each subnet as the vendor instructed. No firewalls were used to protect each subnet. Intrusion detection sensors were connected on interface of the PC in order to monitor traffic. If the system supported stealth operation, it was enabled. On a second interface was connected a management console. These interfaces were aggregated on a separate network to ensure communication with the sensors even under heavy network loads. There were many intrusion detection systems placed under multiple subnets connected to a router and an open firewall in order to test management features [41]. NSS performed five types of tests described below [41]:

- Attack recognition: during which a range of exploits and scans were run using various commercial and "underground utilities" such as nmap, targa, netcat, hping, aggressor, nessus and many more including in house programs in C. The attacks used covered' port scans, denial of service, Trojans, web, FTP, SMTP, POP3, ICMP and finger. Attacks were aimed at a variety of machines with different operating systems and were initiated internally to the subnet the intrusion detection system was on with the exception of fragrouter. The problem of a baseline was addressed with the establishment of certain assumptions. First assumption was that the system should be able to detect the attack in the absence of background traffic and IP fragmentation. Of course, all necessary target applications and servers were installed so as to create victim responses and thus aid detection. NSS dealt with the problem of which and how many attacks to launch by selecting a representative sample. The selection criteria were how common, well publicized an attack was

as well as if there were already existing tools which employed it. Future work is done to include attacks from SAN top 20 and/or ICAT top 10 vulnerability lists. This is expected to increase focus on latest, never seen before attacks, which require new information-gathering and analysis techniques.

- Performance under load: these are basic stress tests. For example a simple Back Orifice ping is utilized to send a stream of 10,000 Back Orifice pings. In conjunction, a listening server is installed in order to count the number of pings received. The test compared the number of pings sent with the number of pings detected under high loads [41]. The actual test consisted of a baseline test and repetition of the above procedure for a number of network utilizations. The baseline consisted of the ping test without any background traffic. One would expect that all pings would be detected if there were no background traffic. Subsequently, background traffic was varied. 64-byte packets with valid source and destination IP addresses and ports were sent to evaluate the raw sniffing capability of the intrusion detection system. The test was repeated for network loads of 25, 50, 75 and 100 per cent. For the same rates mixed traffic utilizing a variety of packet sizes and protocols was attempted. Finally, 1514-byte packets were attempted again for the same variety network loads. Traffic generation was achieved by means of Adtech AX/4000 Broadband Test System with 10/100Mbps modules and SmartBits SMB6000 with LAN-3131A 10/100Mbps SmartMetrics and LAN-3310A 10/100/1000Mbps TeraMatrics cards.
- IDS evasion techniques: intrusion detection systems were tested for their resistance to anti-evasion techniques. The baseline was set through IP forwarding common attacks across a router. Then fragrouter was employed in order to attempt evasion techniques including: ordered and out-of-order 8-byte IP fragments, ordered and out-of-order 8-byte IP fragments with duplicate fragments. Additionally, whisker was used to run basic WWW CGI scan of target machines. Some of the attacks used were: URL encoding, fake parameter, premature URL ending and / ./directory insertion [41].
- Stateful operation test: tools such as stick and snot were used to generate false alerts over a range of protocols and valid source and destination addresses. During that period real attacks were launched in order to determine whether intrusion detection systems could still detect real attacks in a flood of false alarms [41].
- Host performance: Network load, CPU and memory utilizations were monitored during high alert attacks in order to estimate impact of the host carrying the intrusion detection system [41].

E. Trade Magazine Evaluation

Finally we mention a trade magazine evaluation performed on seven intrusion detection systems [27]. This test is included here because of the interesting approach the designers of the test took in implementing it. They placed the intrusion detection systems in the production network of an Internet service provider. Normal traffic on the ISP's network served as background traffic for testing. The average network utilization was in the range of 9 to 12 Mbps traveling on nine T1 lines. Instead of generating attacks, the testers deployed four machines, which had old, unpatched versions of Windows 2000 Server, Windows NT 4.0 Server, Red Hat Linux 6.2 and Sun Solaris 2.6. These machines would attract attackers who in turn would generate attack traffic. The article authors reported that the machines were compromised as soon as they were deployed [27].

The intrusion detection systems used covered a wide range in terms of technology. The systems include appliance intrusion detection systems, signature and anomaly detection including open source as well as proprietary systems. Each system had at least one sensor installed in order to monitor traffic and generate alarms. Furthermore, the designers tried to emulate the management methodology in production networks where of the intrusion detection systems sensors report to alerts to management consoles in other subnets [27].

The metrics used by the designers were accuracy, ease of use, and uptime. Uptime was defined as the number of times a system ceased to function in a period of 30 days. Accuracy was the term used for true positive and negatives as opposed to false positives and negatives. However, in this test the definition of what constitutes a detectable attack is different. They considered an attack to be any compromise of any computing resource on the 'protected' network. This is not the same as an attempted attack; if there was no compromise, the intrusion detection system is essentially reporting a vulnerability that does not exist [27]. Ease of use was another metric they used. It was applied to the amount of information supplied by the intrusion detection system for each alert. How easy it was to figure out which alerts required immediate attention was also considered [27]. No explanation was given on the determination of the baseline. However since this was a comparative test, the author conjecture that the designers of the test were not interested in specific measurements only on which system performed best relative to the others [27].

V. CONCLUSION

Reviewing existing intrusion detection system testing and benchmarking tools and methodologies, it becomes clear that

present approaches to comparative intrusion detection testing and evaluation are inadequate. We do not believe that there is ever going to be a comprehensive test nor a single methodology that would examine an intrusion detection system for every attack, verify its abilities and identify all its weaknesses [28]. This in essence would be to categorically solve the problem of intrusion detection. However, it is possible to improve on the existing testing and evaluation practices employed in the field. At present the philosophy is still that the best way to evaluate any intrusion detection algorithm is to use live or recorded real traffic from the site where the algorithm is to be deployed [22]. Although this is a valid component of test and evaluation, we believe that there is a real need for a new public domain test and evaluation methodology that may be used in a more uniform and repeatable manner.

The 1998 and 1999 DARPA approach was a significant effort toward intrusion detection benchmarking. That approach attempted to provide a realistic evaluation environment. However, in retrospect, it has inadequacies and is not really a good environment. The traffic traces generated from that effort and later published required a lot of rework in order to be reused by others. In addition, the statistical models used to create the background traffic were approximations that do not generalize well.

Recently there has been increased activity in the development of attack generation tools for intrusion detection system testing. Along with that development has been the creation of tools to be used to assist with the analysis of the evaluation results [13-23]. We think that the initial benchmarking of intrusion detection algorithms will move away from placement on network segments with live actual network traffic to an all-one-environmental approach once sophisticated enough tools exit. An example of a recent attempt at this is Thor [48]. Thor automatically launches attacks and collects the alarms that the intrusion detection system generates. Thor has the ability to vary the attacks so as to try to evade the intrusion detection system. This new methodology has many obvious advantages over the traditional methodologies including repeatability and common test metrics that may be applied by a wide range of intrusion detection implementers and evaluators.

A new and more capable open source evaluation methodology is needed in the intrusion detection community. This new methodology must be able to generate realistic network background traffic through artificial generation as well as merging capability with existing or live network traffic. The synthetic component of background traffic must be generated as realistic traffic streams that included full payloads. Typical network simulators like *ns* do not at present incorporate the ability to generate realistic data from the standpoint of full traffic stream generation, thus the most popular open source network simulation tool will require major enhancements if it is to be used as the basis for traffic generation. In addition, the massive amounts of traffic that must be generated require a parallel and distributed simulator in order to be able to generate the large amounts of traffic required. The ability to merge in existing traffic trace files either from previous generation runs or actual traffic capture files is also required. This allows the continued use of live network traffic which to date is still the most prevalent test methodology. This proposed new methodology requires a strictly controlled and repeatable component and yet also requires a live environment component capability. The live environment component must include the ability to inject attacks in the live traffic component so that a library of attacks may be injected along with the live traffic component. For obvious reasons it is not desirable to inject these library attacks into the actual live network traffic segment, injection must occur in an isolated test environment that includes the live traffic component. The capability of the proposed test environment to generate and respond to traffic in real time will allow the new methodology to be inserted into existing test beds. The proposed environment should work simultaneously with real network segments as well as captured network segment traffic. The background traffic generator will require the ability to parse existing traffic traces and then merge additional background or attack traffic into repeatable trace files. All of this must occur in real time since intrusion detection algorithms may adapt or provide feedback into the network environment. The traffic generation capabilities must include gigabit traffic generation capability. Realistic user models and network activity must be incorporated. The ability to use actual machines along with simulated machines is also required. As an example, Linux based machines that act as multiple independent traffic generators already exist. Servers offering actual services should be able to be placed into the proposed environment. The ability to insert real platforms with standard operating systems is a requirement since the only way to see vulnerabilities is to use the actual system under stressful environments.

Attacks should be maintained in an attack repository such that they may be inserted into the background traffic activity. Standard attack and evaluation tools that already exist should be a part of this attack repository, and they should be able to execute in the same manner that the tools execute when run in a traditional stand-alone configuration. An automated method for managing attacks is also required. This automated method would have the ability to inject the attacks at specific times or as a result of specific network conditions. For each attack executed, the environment should have the ability to report the reactions of the intrusion detection algorithms.

The tools and methodologies needed for uniform testing of intrusion detection systems do not yet exist in the public domain at a level necessary for performing the necessary comparative tests and evaluations of new algorithms. There is a great need for an open source traffic generation and attack insertion environment that may be uniformly used for intrusion detection evaluations. The end result of a new methodology having the capabilities we have proposed would be a uniform test and evaluation capability that could be used to make

meaningful comparisons between various intrusion detection algorithms and systems. At present, this does capability does not exist.

REFERENCES

[1] Abily, V. and Ducasse, M., "Benchmarking a distributed intrusion detection system based on ASAX: Preliminary results" Extended abstract presented at RAID 2000.

[2] Zhnag, K. "A Methodology for Testing Intrusion Detection Systems," M.S. Thesis, University of California at Davis, May 1993.

[3] Champion, Terrence and Denz, Mary L., "A Benchmark Evaluation Network Intrusion Detection Systems", IEEE 2001.

[4] Champion, Terrence G. and Durst, Robert S. "Air Force Intrusion Detection System Evaluation", www.raid-symposiom.org

[5] Cunningham, R. K.; Lippmann, R. P.; Fried, D. J.; Garfinkel, S. L.; Graf, I., Kendal, K. R.; Webster, S. E.; Wyschogrod, D. and Zissman, M. A. "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation", In Proceedings of the Third Conference and Workshop on Intrusion Detection and Response (SANS 1999).

[6] Kendall, K., "A Database of Computer Attacks for the Evaluations of Intrusion Detection Systems", M.S. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1999.

[7] Korba, J., "Windows NT Attacks for the Evaluation of Intrusion Detection Systems" M. S. Thesis, MIT Department of Electrical Engineering and Computer Science, June 2000.

[8] Lippman, Richard; Haines, Joshua W.; Fried, David J. and Korba, Jonathan, "The 1999 DARPA Off-Line Intrusion Detection Evaluation"

[9] Lippman, Richard; Haines, Joshua W.; Fried, David J.; Korba, Jonathan and Das, K., "The 1999 DARPA Off-Line Intrusion Detection Evaluation", Computer Network, 34(4):579-595.

[10] Lippmann, Richard P.; Cunnigham, Robert K.; Fried, David J.; Graf, Isaac; Kendall, Kris R.; Webster, Seth E.; Zissman, Marc A. "Results of the DARPA 1998 Offline Intrusion Detection Evaluation", www.ll.mit.edu/IST/pubs.

[11] Lippmann, Richard P.; Fried, David J.; Gaf, Isaac; Haines, Joshua W.; Kendall, Hristopher R.; McClung, David; Weber, Dan; Webster, Seth E; Wyschogrod, Day; Cunningham, Robert K. and Zissman, Marc A., "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation", In Proceeding of the DARPA Information Survivability Conference and Exposition: DISCEX-200,.Volume 2. Los Alamitos, California: IEE Computer Society , 25-27 January 2000, pp. 12-26, Hilton Head Island, South Carolina.

[12] Lippmann, Richard P.; Graf, Isaac; Wyschogrod, Dan; Webster, Seth E.; Weber, Dan J. and Gorton, Sam, "The 1998 DARPA/AFRL Off-Line Intrusion Detection Evaluation," *First International Workshop on Recent Advances in Intrusion Detection (RAID)*, Louvain-la-Neuve, Belgium, 1998.

[13] Maxion, Roy A. and Tan, Kymie M.C., "Benchmarking Anomaly-Based Detection Systems", 1st International conference on Dependable Systems & Networks. New York, New York, USA: IEEE, 25-28 June 2000, pp.623-630.

[14] Maxion, Roy A., "Measuring Intrusion Detection Systems. Web proceedings of the First International Workshop on Recent Advances in Intrusion Detection (RAID'98), www.raid-symposium.org/raid98

[15] McHugh, John, "Testing Intrusion Detection Systems: A Critique of the 1998 and 199 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory", ACM Transactions on Information and System Security, Vol. 3, no. 4, November 2000, pp. 262-294.

[16] Me, Ludovic and Michel, Cedric, "Intrusion Detection: A Bibliography", Technical Report SSIR-2001-01, September, 2001.

[17] Mell, P. "Acquiring and Deploying Intrusion detection Systems", National Institute of Standards and technology's Information Technology Laboratory bulletin.

[18] Puketza, Nicholas J.; Chung, Mandy; Olsson, Ronald A and Mukherjee, Biswanath, "A Software Platform for Testing Intrusion Detection Systems", IEEE Software, September/October, 1997, 43-51.

[19] Puketza, Nicholas J.; Zhang, Kui; Chung, Mandy; Mukherjee, Biswanath and Olsson, Ronald A., "A methodology for Testing Intrusion Detection Systems" 17thNatrional Computer Security Conference: Baltimore, MD, October, 1994.

[20] Puketza, Nicholas J.; Zhang, Kui; Mukherjee, Biswanath and Olsson, Ronald A., "Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype" Proc. 17thNatrional Computer Security Conference, Vol. 1, pp.1-10, October, 1994.

[21] Puketza, Nicholas J.; Zhang, Kui; Chung, Mandy; Mukherjee, Biswanath and Olsson, Ronald A., "A methodology for Testing Intrusion Detection Systems", IEEE Transactions on Software Engineering, 22, 1996, pp. 719-729.

[22] Ranum, Marcus J., "Experiences Benchmarking Intrusion detection Systems", www.nfr.com, 5 May 2002.

[23] Rossey, Lee M.; Rabek, Jesse C.; Cunningham, Robert K.; Fried, David J.; Lippmann, Rich P. and Zissman, Marc A. " LARIAT: Lincoln Adaptive Real-time Information Assurance Testbed", presentation in RAID 2001, 10 October 2001.

[24] http://www.der-keiler.de/Mailing-Lists/securityfocus/focus-ID system/2002-01/0081.html (7/2/2002)

[25] http://www.antara.net/home.html (7/2/2002)

[26] Haines, Joshua W., Rossey, Lee M., Lippman, Richard P., Cunningham, Robert K., "Extending the DARPA Off-Line Intrusion Detection Evaluations", In the Proceedings of DISCEX 2001, June 11-12, Anaheim, CA.

[27] http://www.nwfusion.com/techinsider/2002/0624security1.html (7/1/2002)

[28] Loscocco, Peter A, ; Smalley, Stephen D.; Muckelbauer, Patrick A.; Taylor, Ruth C.; Turner; S. Jeff and Farell, John F. "The inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments", National Security Agency

[29] http://www.hping.org/ (7/5/2002)

[30] Ptacek, Thomas H. and Newsham, Timothy N. " Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html#cit6 (7/5/2002)

[31] http://securityresponse.symantec.com/avcenter/security/Content/2000_06_01.html (7/5/2002)

[32] http://www.roland-riegel.de/nload/index_en.html (7/5/2002)

[33] www.netpref.org (7/5/2002)

[34] http://dast.nlanr.net/Projects/Iperf/ (7/5/2002)

[35] http://www.webattack.com (7/6/2002)

[36] http://www.hackingexposed.com/tools/tools.html (7/6/2002)

[37] http://www.physnet.uni-hamburg.de/physnet/security/vulnerability/synk4.html (7/6/2002)

[38] http://packetstormsecurity.nl/UNIX/IDS/nidsbench/nidsbench.html (7/6/2002)

[39] http://www.cawnetworks.com/product/index.shtml (7/6/2002)

[40] http://www.shmoo.com/mail/ids/oct99/msg00474.html (6/30/2002)

[41] "Intrusion Detection Systems: Group Test (Edition 2)" The NSS Group, Oakwood House, Wennington, Cambridge, PE28 2LX, England, UK: www.nss.co.uk (7/12/2002)

[42] http://www.shmoo.com/cctf/ (7/16/2002)

[43] http://www.hsc.fr/ressources/outils/idswakeup/index.html.en (7/16/2002)

[44] http://archives.neohapsis.com/archives/sf/ids/2001-q2/0316.html (7/16/2002)

[45] http://cebu.mozcom.com/riker/iptraf/about.html (7/16/2002)

[46] Das, J. Kumar "Attack Development for Intrusion Detection Evaluation " M. S. Thesis, MIT Department of Electrical Engineering and Computer Science, June 2000.

[47] http://tcpreplay.sourceforge.net/

[48] [Marty, Raffael, "THOR: A Tool to Test Intrusion Detection Systems by Variation of Attacks", Diploma Thesis Swiss Federal Institute of Technology Zurich, March 2002.

[49] http://www.insecure.org/tools.html (7/2/2002