WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2015; **15**:1851–1864
Published online 24 February 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/wcm.2471

RESEARCH ARTICLE

Real traffic logs creation for testing intrusion detection systems

Wassim El-Hajj[1]*, Mustafa Al-Tamimi[1] and Fadi Aloul[2]

[1] Computer Science Department, American University of Beirut, Beirut, Lebanon
[2] Computer Science and Engineering Department, American University of Sharjah, Sharjah, UAE

ABSTRACT

Port scanning is one of the most popular reconnaissance techniques that many attackers use to profile running services on a potential target before launching an attack. Many port scanning detection mechanisms have been suggested in literature. To test the proposed detection approaches, researchers use data sets that are available online or simulate their own. However, the available data sets do not provide complete logs and are usually outdated. Furthermore, the simulated data sets provide logs that do not resemble real-life scenarios. These deficiencies in the available data sets highly affect the performance of testing the intrusion detection systems (IDSs) and result in poor evaluations. Meanwhile, very little work has been done on generating port scanning benchmarks that researchers can use to test their detection methods. In this work, we suggest a simulation framework using OMNeT++ to generate benchmarks that resemble real-life traffic. We approach the problem by dividing it into three modules: (1) topology creation; (2) good traffic generation; and (3) bad traffic generation, each of which are made realistic, similar to deployed and usable networks. The benchmark is then tested using Snort and MalwareAnalysis. The tested IDSs were not able to catch many of the generated port scanning attacks, specifically the slow and distributed ones. We also measured the attack detection efficiency of the IDSs under different loads of background activities. Hence, the proposed framework and the annotated benchmarks will provide researchers and industry with an effective way of testing the power of IDSs' port scanning detection modules. Copyright © 2014 John Wiley & Sons, Ltd.

KEYWORDS
benchmark testing; computer security; intrusion detection; simulation

***Correspondence**
Wassim El-Hajj, Computer Science Department, American University of Beirut, Beirut, Lebanon.
E-mail: we07@aub.edu.lb

1. INTRODUCTION

In networking, new protocols and systems are usually tested, at least in the early stages, using simulations as opposed to testing and troubleshooting using real systems [1]. Only when the simulation results are satisfactory, implementation on real hardware is performed. Compared with the cost and efforts spent in establishing a real testbed environment, network simulators have proven to be relatively fast, accurate, and inexpensive regardless of the protocol being simulated or its layer [2]. Typical systems that are tested via simulation include intrusion detection systems (IDSs), which are used as a first line of defense against intrusions.

Port scanning is one of the major reconnaissance techniques that is used by the hackers prior to launching an attack. It is a targeted form of information gathering that attempts to profile the services that are running on a potential target by probing the target for open ports [3]. While profiling the services, the attacker's main aim is to discover services with weak security or well-known vulnerability. Finding such a service allows attackers to perform malicious activities ranging from *passive* attacks such as extracting secure information to *active* attacks such as implanting viruses, worms, and Trojan horses in the network. Although a scan is not an attack and does not harm its targets, it is considered a highly invasive activity that uncovers security loopholes and leads hackers to launch successful attacks [4]. Therefore, detecting port scanning activities at early stages will result in reducing security breaches.

Intrusion detection systems are mainly designed to gather and analyze information over a network and detect any possible security breach such as port scanning and other anomalous activities. Although IDSs do protect the corporate network from many intrusion attempts, malicious users are continuously finding new ways to bypass

the IDS and get access into the internal network [3,4]. When used smartly, the activity of port scanning is an example of such malicious actions that can deceive the IDS and go unnoticed.

Because of the dangerous consequences of port scanning, every IDS comes with a port scanning detection module based on one of the following approaches: time-based [5], connection-based [6], and machine learning [7]. However, by devising smart port scanning activities such as *slow* port scanning or *distributed* port scanning, adversaries can still bypass the deployed IDSs, making this field a challenging area for researches [8]. As a result, network administrators must perform rigorous assessments and tests to make sure that the IDSs protecting the corporates do provide promising results as claimed by the IDS vendors. Consequently, it is of vital importance to develop a benchmark that enables vendors as well as network administrators and researchers to effectively evaluate and test IDSs.

The main aim of testing IDSs is to evaluate the *hit rate* and *false alarm* ratio. The hit rate ratio determines the level of correctly detected attacks, while the false alarm ratio indicates the wrong alarms produced by the IDS. Researchers consistently test their detection approaches by using data sets that are available online [9]. A major drawback of the available data sets is that they are outdated and do not contain specific information regarding the number of attacks and their types. These deficiencies in the available data sets highly affect the performance of testing the IDSs and results in poor evaluations. One such data set is KDD'99 [10], which was distributed since October 1999 by the University of California, Irvine. Currently, researchers perform these tests under simulated environments, because of the high costs and risks involved in testing under real environments [11]. The quality of simulation results significantly depends on the simulation environments that should resemble realistic behaviors. Meanwhile, successful and effective evaluation of IDSs requires overcoming several challenges summarized hereafter. One of the challenging tasks in testing IDSs is collecting attack scripts. Although many attack scripts exist online, it takes a considerable amount of time and effort to adapt them in simulation. Moreover, these scripts are produced by different developers to work in different environments, hence adding more complexity when integrating them into the simulated environment [11]. Another challenging task that is crucial in IDS testing is the generation of background traffic that will be further discussed in Section 2.

In this work, we develop a comprehensive framework that generates benchmarks for port scanning testing. Our proposed approach tackles all the hurdles in testing an IDS and provides a realistic simulation environment (ReaSE) that consists of three stages. We first create a topology that can be easily extended to include normal as well as malicious users. We then create the modules that simulate real traffic and show via simulations that our generated normal traffic resembles self-similar traffic and follows real traffic patterns. Finally, we create the modules that generate the port scanning attacks. The modules were designed and implemented using the discrete event simulator OMNeT++ [2], which we believe can be used as a base simulator for testing IDSs, because of its considerable functionality and ease of use. It is to be noted that little work has been done on developing benchmarks for port scanning. Most researchers use for their testing, data from log files that do not contain many port scanning activities or contain manually generated port scanning traffic [3]. The rest of this work is structured as follows: Section 2 briefly overviews port scanning, challenges in testing IDSs, the used simulator OMNeT++, and finally a summary of related work. Section 3 explains our approach for creating the benchmark by discussion of topology, background traffic, and attack traffic. In Section 4, we present the results of testing our benchmark. Section 5 concludes the paper and presents the future work.

2. BACKGROUND AND RELATED WORK

The purpose of this section is to provide the reader with some background information in port scanning, IDSs, and the simulation environment. We first explain the activity of port scanning and the most used techniques by hackers. Later, we mention the challenges in providing proper background traffic to test the IDSs and explain briefly the different types of IDSs. Next, we highlight our simulation environment: OMNeT++. Finally, the recent related work is discussed.

2.1. Port scanning

Port scanning is one of the most common and considerable techniques for discovering an open door (port) in a system that allows the intruder to launch malicious attacks through the discovered port. In this method, the adversary may use different types of port scanning approaches, such as SOCKS port probe, stealth scan, bounce scan, Transmission Control Protocol (TCP) scan, and User Datagram Protocol (UDP) scan [3]. The detection methodologies to these types of scans have been the main focus for IDSs, because prevention of such activities is the most crucial step to stop an attack. Although several approaches were proposed to detect port scanning attacks, the adversaries have been developing new techniques to avoid the detection of their activities, therefore, making this field an important area for security researchers.

In order to generate the attack traffic, first, we take an insight into the port scanning activity. Basically, there are 65 536 standardly defined ports on a computer that are classified into three ranges: (1) well-known ports (0–1023); (2) registered ports (1024–49151); and (3) dynamic/private ports (49152–65536) [12]. Computers connected to a network exploit many services that use TCP/UDP protocols by connecting through these ports. Essentially, a port scanner sends a message to each port and waits for a certain

response. Depending on the received response, the port scanner can discover whether the port is closed or being used and further continue discovering the weakness to exploiting the offered service. Most of the port scanning activities use TCP that is based on a connection-oriented protocol and provide scanners with trustworthy results. However, some of the port scan activities may use UDP that is based on a connectionless service. The drawback of using UDP is that it can be easily blocked by firewalls and may not return consistent information because of the connectionless type of services [12].

A TCP connection is established by a three-way handshake that is explained in Figure 1, and the listening application (server) is informed only when the handshake is successful. When a user initiates a connection, it first sends a TCP packet that carries an Synchronous (SYN) flag. If the port is open on the server side, then it will respond with a TCP packet containing the Synchronous/Acknowledgment (SYN/ACK) flag after which the initiating user will respond with a TCP ACK message, and finally, the connection is established. In case the port is closed, the server will reply with a TCP containing Reset (RST) flag [13].

The following are some of the mostly used and well-known port scanning attacks that use TCP [3]. They are also summarized in Table I.

- *Connect scan:* A TCP Connect scan completes the three-way handshake, and after a successful attempt, it is logged as a connection. It is the easiest type of attack to perform because it does not require root privileges. The port is considered open when the connection is established and closed otherwise. If the connection is successful, the attacker sends a Finish (FIN) packet to end the connection. This type of scan is usually recorded in the log file of the server.
- *SYN scan:* It is considered the most popular type of port scanning and usually referred as TCP half Connect scan. The scanner initiates by sending a SYN packet, and after receiving SYN/ACK (open port), the attacker responds with RST in order not to complete the three-way handshake. Therefore, the scan does not show up in the application level logs because the connection is not established. This process is also referred to as stealth scan given that it is usually not recorded in the log files.
- *FIN scan:* This type of attack is used when the network firewall drops all SYN–ACK packets to the closed ports. The firewall however allows all inbound packets with FIN; hence, the scanner sends a FIN packet to the destination, and upon receiving an RST response, it means the port is closed. If the port is open after sending a FIN, there will be no response. This process is usually not recorded in the log files of the server.
- *ACK scan:* Whenever a scanner wants to determine filtered ports by the firewall, it initiates an ACK scan. If the response is RST, then the remote port is unfiltered and is accessible. However, when there is no response, this indicates that the port is filtered by the firewall. This process is usually not recorded in the log files of the server.

Port scanning can be classified into two broad categories: single-source (one-to-many) port scans and distributed (many-to-many or many-to-one) port scans. Distributed port scanning is the kind of scanning where the intruder launches a coordinated scan distributed among

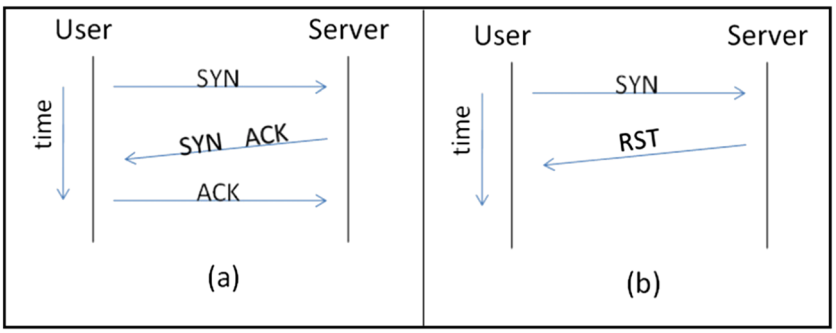


Figure 1. Transmission Control Protocol three-way handshake. (a) The three-way handshake with an open port. (b) Connection attempt on a closed port.

Table I. Summary of the most popular port scanning attacks.

Scan technique	TCP flag	Response open	Response close	Firewall detection
Connect scan	SYN	ACK	RST	Yes
SYN scan	SYN	ACK	RST	No
FIN scan	FIN	NO	RST	No
ACK scan	ACK	NO	RST	No

multiple parties. Each party selects a set of the available ports and scans them. The attacker then collects the scan results from every party. Such an attack has many similarities with distributed denial-of-service attacks that have long been a challenge to the research community [12].

Next we briefly explain the IDSs and the difficulties in testing them under different background activities.

2.2. Intrusion detection systems

Intrusion detection systems are designed specifically to gather and analyze information over a network and detect any possible security breaches [1,14,15]. IDSs are usually used as a first line of defense against intrusions. Every commercial IDS comes with a port scanning detection module based on one of the following approaches: time-based [3], connection-based [4], and machine learning [5].

In terms of detection approaches, IDSs are classified into three categories: signature-based, anomaly-based, and specification-based systems. The signature-based detection, also known as the misuse-based detection, attempts to recognize attacks that follow certain intrusion patterns. Such patterns are typically collected from previously known attacks. However, this approach is ineffective against previously unseen attacks. On the other hand, anomaly detection can be identified by recording unusual behavior of operations. If such operations are sufficiently different from known normal behaviors, then they are considered as abnormal traffic hence as assaults. This method can detect unknown attacks yet can produce false alarms for legitimate but previously unseen behaviors.

Specification-based detection combines the strength of signature-based (accurately detecting known attacks) and anomaly-based (detecting unknown) by manually specifying program behavior specification. A hybrid lightweight IDS was suggested in [16].

The main aim of testing IDSs is to evaluate the hit rate and false alarm ratio comprehensively. The hit rate determines the level of correctly detected attacks, while false alarm ratios indicate the mistaken alarms produced by the IDS. Most IDS testing approaches can be classified into one of the four environments depending on the background traffic generation patterns:

(1) *No background traffic*: In this approach, the IDS is deployed to only capture and analyze the attack scripts without any background activity. The evaluation metrics used are attack detection accuracy and hit rate precision. However, in this approach, it is impossible to evaluate neither the false alarms nor the robustness of the IDS under high levels of background activities [17].
(2) *Real background traffic*: Testing in real environment with realistic background traffic is very effective in finding the hit rate and false alarm ratio because of the background activity. However, it is impossible to guarantee the identification of all attacks because there is no prior knowledge about hidden ones. Another major drawback of such approach is that the data cannot be distributed because of privacy concerns [18].
(3) *Sanitized background traffic*: In sanitization, all the sensitive data are removed from a realistic traffic log, in order for it to be distributed and analyzed freely without any concerns regarding privacy issues. After sanitization, attack traffic is injected for testing the IDS. However, this sanitization may remove essential information that is needed for the attack detection and might rule the traffic as unrealistic [19].
(4) *Generated background traffic*: In this approach, the background traffic is generated by complex traffic generators that model realistic traffic behavior [20]. Because it is guaranteed that the generated traffic does not contain any unknown attack, it yields to a precise evaluation of false alarms and hit rate ratio. One more advantage of this approach is that the tests can be repeated by generating the same traffic again. However, the main challenge remains: to make sure that the generated traffic resembles realistic scenarios.

In this work, we adopt the last environment and develop a comprehensive framework that generates benchmarks for port scanning testing. The generated traffic between hosts resembles realistic traffic patterns that will yield meaningful and accurate evaluation results. The details of traffic generation are further explained in Section 3. We next highlight the network simulator OMNeT++, which is used in our simulations.

2.3. OMNeT++ simulator

OMNeT++ is a discrete event simulator that is based on hierarchical nested modules. These modules communicate using messages through channels. Simple modules lie at the lowest level of the hierarchy and combine one or more C++ classes that describe the algorithm and functionality. Compound modules consist of one or more simple modules and define the interconnection among them. Moreover, the compound modules can be interconnected through incoming and outgoing gates called channels. OMNeT++ models are often called systems that are at the top level of the hierarchy, and each system contains simple and/or compound modules where each compound module consists of multiple simple modules as shown in Figure 2.

Each simulation program consists of the following four components:

- *Simple module source* that is implemented in C++ using the OMNeT++ simulation class library. These source files contain the algorithm of the simple module and have (.cc) and (.h) suffixes.
- *Network description* that specifies the topology of module connections using the NEtwork Description (NED) language and is saved in specific files with

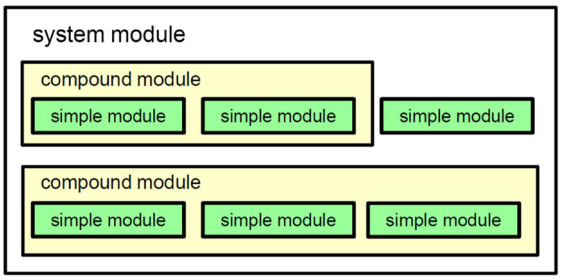


Figure 2. OMNeT++ model structure.

(.ned) suffix. This file sets the structure and parameters of simple modules, compound modules, channels, connections, and network definitions.

- *Messages* is an instance of cMessage class of OMNeT++ that represents events, messages, packets, or other entities in a simulation. It is created using the message class in OMNeT++ with (.msg) suffix, and it is sent from one module to another with specific types and structure.
- *Initialization file* with (.ini) suffix contains the general settings for the execution of the simulation. It also sets values to the parameters included in the NED file that will specify the traffic type and simulation behavior.

OMNeT++ offers variety of services for specialized areas. The INET framework is amongst the very well-known extensions that support simulations of the common Internet protocols such as TCP, Internet Protocol (IP), UDP, and Internet Control Message Protocol (ICMP) along with other services such as Packet Capturing. The details of each implemented protocol can be found in the corresponding Request for Comment document [21].

2.4. Related work

In recent studies, researchers have proposed several ways to test IDSs under various environments such as real background traffic, no background traffic, or generated background traffic. For real background traffic, different data sets have been used for testing purposes. The authors in [22] publicly released internal enterprise traffic, known as Lawrence Berkeley National Laboratory data sets (LBNL) that span more than 100 h of activity over a total of several thousand internal hosts. The main applications observed were web, mail, and name services. Meanwhile, the traffic was anonymized to ensure that user privacy is preserved. The LBNL attack traffic mostly consists of malicious nodes that perform TCP port scans, which are targeted at LBNL hosts. In [3], a data set known as endpoint background traffic was presented. The data set consists of data exchanged in home and university environments. Because home computers are usually shared among multiple users and run peer-to-peer applications, they generate significantly higher traffic volume than university computers. The attack traffic was mostly composed of outgoing port scans, as opposed to LBNL traffic where attack traffic is inbound. In [23], the authors gathered real traffic traces from a dormitory at Seoul National University to test their proposed port scanning detection method. They used a fast increase slow decrease scheme that automatically and adaptively sets the port scanning detection threshold, on the basis of traffic statistical data during prior time periods. The proposed method outperformed Snort [5], BRO [24], and Threshold Random Walk.

In [25], the authors created a port scanning testbed using Defense Technology Experimental Research network that uses EMULAB software [26]. DScan [27] and NSAT [28] scanner tools were then used to perform distributed port scanning activities. The testbed was used to evaluate the proposed detection algorithm that is based on solutions to the set covering problem [29]. The algorithm successfully detected strobe scan (scanning multiple ports on multiple hosts) and horizontal scans (scanning a specific port on multiple hosts) against contiguous address space. In [2], the authors created their own port scanning benchmark in OMNeT++ using uniform distribution of traffic. The data was used to test the proposed Voice over Internet Protocol IDS. However, the generated traffic did not convey a realistic behavior to test the false alarm ratio.

In [30], the authors built a model called ReaSE that is developed on top of INET framework (an extension of OMNeT++), hence inheriting the advantages of simulating the Internet protocols such as TCP/IP. ReaSE creates a ReaSE by considering multiple aspects of network simulations such as topology generation and realistic traffic patterns. The generated traffic resembles realistic traffic and exhibits self-similar behavior. However, ReaSE is not being updated to be compatible with the recent INET framework that supports recent protocols [31]. In our approach, we borrow some of the concepts from ReaSE along with hand crafted concepts to create and incorporate extra modules into INET framework to be able to generate realistic traffic while embedding port scanning activities within the generated traffic.

3. BENCHMARK FRAMEWORK COMPONENTS

In network evaluation, a standard simulation must define the topology of the network and traffic patterns of the simulated hosts along with anomalous nodes. In this section, we detail our approach to developing port scanning benchmark with realistic logs. First, we explain the suggested network topology and what is the state of the art in generating different topologies. Then, we detail the traffic generation methodology. We end the section by describing the bad traffic (port scanning) generation and analyzing the generated benchmark.

3.1. Network topology

To generate realistic topologies, two approaches are mainly used [30]. The first approach is based on observing real-

life scenarios collected from Border Gateway Protocol (BGP routing data) or RouteViews project [32]. The main drawback of real observations is that the collected data are not easily integrated with the simulator given the heavy load of data. Moreover, the observed data are not updated on the basis of current topologies [30]. The second approach relies on random topology generation. However, random networks do not accurately model real topologies because major parameters such as link metrics and internal BGP configurations are often ignored [33]. Even with this drawback, the research community heavily uses random topology generation that depends on power-low distribution [33]. In this random approach, few nodes contain lots of edges resembling the core network, while the rest of the nodes have few edges resembling hosts and routers. The communicating links between nodes have different parametric values to resemble realistic network topology. For instance, the link bandwidth between core and gateway is larger than the link bandwidth between gateway and edges. Meanwhile, the connectivity decreases from edges along the cores. In OMNeT++, the INET framework is capable of generating such random topologies. However, it does not take into account the realistic parametric values in link speeds and routers. Another option for topology generation is BRITE [34], which can be integrated with NS-2 and OMNeT++. In summary, most existing random topology generators are based on Positive-Feedback Preference that randomly implements power-low distribution to the nodes [35].

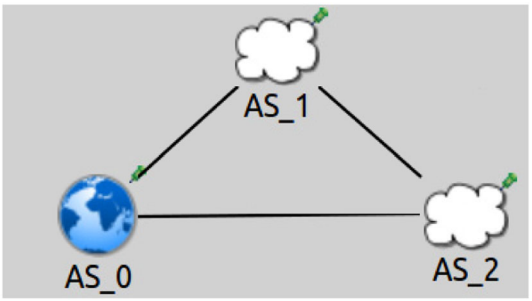


Figure 3. Network topology connecting Autonomous Systems.

Generation of realistic topology is based on Autonomous System (AS) (a network under a single administrative authority) that includes single or multiple domains. Under each AS, the router-level topology or subnets can be further specified. In ReaSE, the generation of realistic topologies is divided into two parts because of the hierarchical structure of the Internet. First, the connection of AS level with multiple separate domains is established as shown in Figure 3. Second, generation of router-level topology under each AS is performed. The router-level topology varies from a two-tiered to three-tiered canonical structure based on the requirements of a domain. The three-tiered structure consists of core, gateway (aggregate), and edge routers as shown in Figure 4. In a two-tiered structure, the core and gateway routers are collapsed in one tier and hence a two-tiered structure consists of core and edge routers only. This method is based on Positive-Feedback Preference that randomly implements power-low distribution to the nodes [12]. The Internet can be seen as connecting ASs together with different specifications [35].

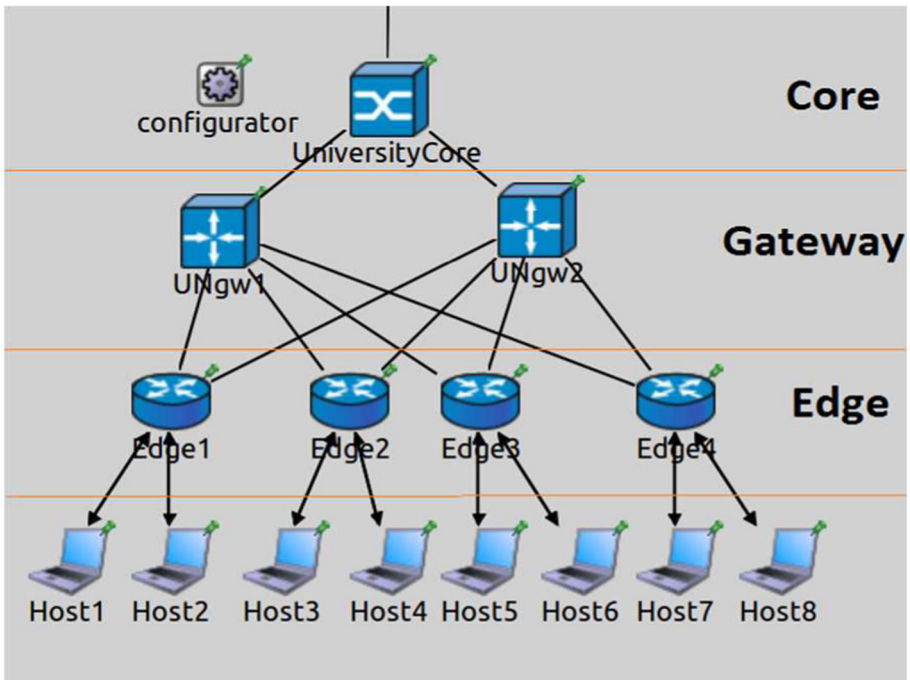


Figure 4. Router-level topology with a canonical three-tier structure (core, gateway, and edge).

Table II. Links properties in the proposed topology.

Router level	Link speed	Router level
Core	<- - - - 40 Gbps - - ->	Core
Core	<- - - - - 20 Gbps - - - ->	Gateway
Gateway	<- - - 10 Gbps - - ->	Edge
Edge	<- - - - 2.5 Gbps - - - ->	Server
Edge	- - 100 Mbps - - ->	Host
Host	- - 100 Mbps - - ->	Edge

The cores that represent the largest and most capable routers are all connected to each other through very high-speed links, especially because they are used to connect together the various ASs. Furthermore, each core connects to few gateway routers via high-speed links (for instance, in Figure 4, UniversityCore connects to University gateways, UNgw1 and UNgw2) and each gateway connects to multiple edge routers. Finally, the edges connect to several hosts with lower speed links. Figure 4 represents a canonical three-tiered router-level topology, and Table II presents the chosen parameters for link metrics. The second row of Table II (Core< - - - - 40 Gbps - - - >Core) can be read as follows: bi-directional links with a speed of 40 Gbps connect core routers. The last row means that hosts connect to edge routers via unidirectional links with a speed of 100 Mbps. It is to be noted that, in the year 2000, the speed of the links connecting core routers was around 2.5 Gbps and this number increased to 10 Gbps a few years later. As of the year 2012, the typical link speed connecting core routers became 40 Gbps [36].

In our simulation environment, we have adopted one of the random topologies generated by ReaSE and modified the nodes and links characteristics to fit realistic networks. This environment includes four ASs that resemble *University*, *Private Enterprise*, *Cloud Data Centers*, and *Peer-to-Peer* networks. The specifications of each AS have been gathered from the studies conducted by Benson *et al*. [35] and follow the two-tier and three-tier structures. We connect these four ASs in such a way that all the external attacks are launched from the Peer-to-Peer AS to the three other ASs. On the other hand, each of the Private Enterprise and University ASs includes inbound attacks from their internal domains. The canonical structure of the University AS is a two-tiered topology while the Cloud Data center and Private Enterprise are three-tiered structures. Figure 5 presents a high level view of the proposed topology connecting the four ASs, while Figure 6 presents the more detailed topology. The Peer-to-Peer AS consists of two scanners that perform fast scan and tricky scan, respectively, three slow scanning hosts that launch slow scans using different timing pattern, and four hosts that launch subtle slow and distributed port scans. The details of each attack along with the scenarios will be explained in Section 3.3 of this paper. Moreover, there are two arrays of normal users (j and k) that produce realistic traffic and connect to Cloud, University, and Private Enterprise. The parameters j and k vary depending on the load of traffic we plan to generate so that we can test the performance of the IDSs under stressful conditions. Hence, the attacking nodes also generate normal traffic in addition to the attack traffic.

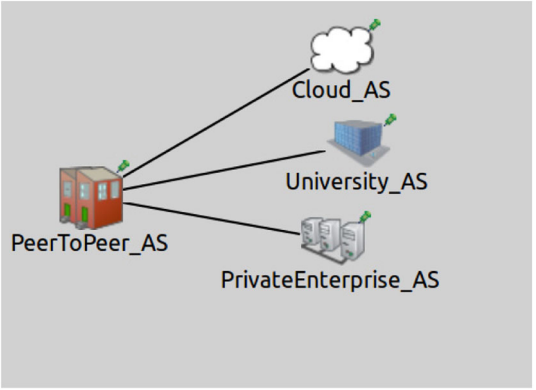


Figure 5. Connecting all Autonomous Systems (Peer to Peer, University, Cloud, and Private Enterprise).

3.2. Traffic generation

Having created the appropriate topology, it is important to make sure that the traffic generation between hosts resembles realistic traffic patterns in order to get meaningful and accurate evaluation results. Creating such patterns requires the generation of self-similar behavior [25], which is based on a reasonable combination of multiple kinds of traffic. The following tools are among the well-known traffic generators that can produce self-similar traffic patterns: ReaSE [30], BonnTraffic [20], TrafGen [13], and D-ITG [37]. One possibility to achieve self-similar traffic behavior is to use multiple traffic sources that are switched on and off on the basis of heavy-tailed intervals [38]. Another possibility is to produce traffic at packet level by replicating appropriate stochastic processes for both inter-departure time (IDT) and packet size random variables (exponential, uniform, Cauchy, normal, Pareto, etc.) [37].

Many studies have been done in analyzing the behavior of network traffic [39,40]. Recently, new models of generating network traffic have been developed [41] (Pareto, Weibull) that are different from the traditional traffic models such as Poisson and Markov. Poisson is a discrete probability distribution that was used in modeling the telephone networks, and because of its good approximation in telephony systems, some researchers adopted this method to produce network traffic until Pareto or Weibull distributions were introduced. Pareto is a power law probability distribution that is named after the famous Italian economist Vilfredo Pareto. Nowadays, self-similarity [25] is the most well-known and used model to generate network traffic that exhibits burst behavior in different time scales in contrast to the smooth model of Poisson distribution.

According to [39] and [40], the definition of self-similarity is referred to standard time series of having X to be a covariance stationary stochastic process where

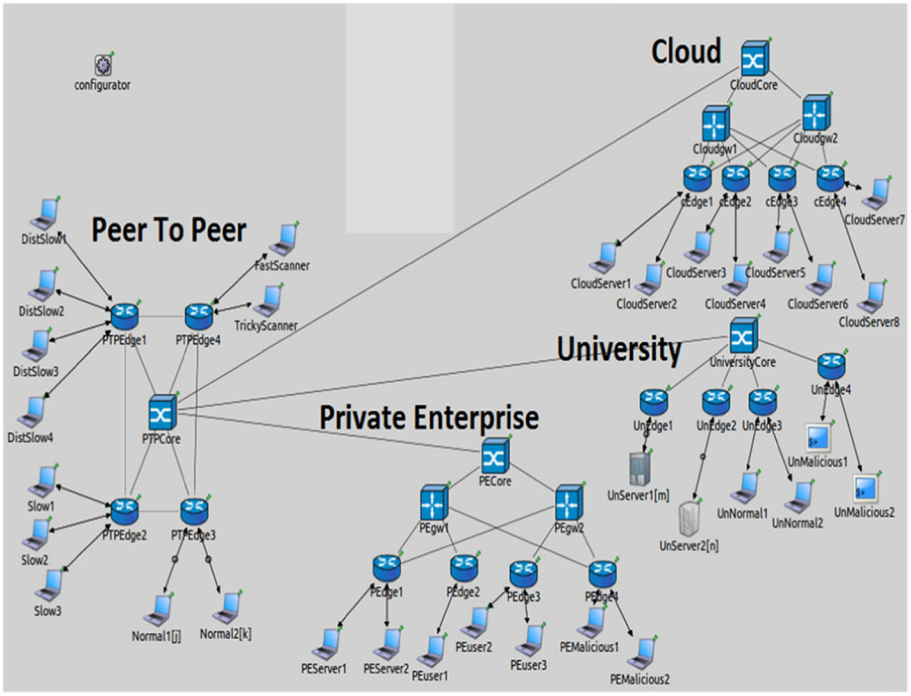


Figure 6. Proposed topology in details.

$$(X_t, t = 0, 1, 2, .., N) \tag{1}$$

where N is the number of observations.

With mean μ and finite variance σ^2: ($E[x]$ is the expected value of x, also known as mean of x)

$$\mu = E[X_t] \tag{2}$$

$$\sigma^2 = E\left[(X_t - \mu)^2\right] \tag{3}$$

with an auto correlation function $r(k)$ that depends only on k

$$r(k) = \frac{c}{\sigma^2} = \frac{E[(X_t - \mu)(X_{t+k} - \mu)]}{E\left[(X_t - \mu)^2\right]} \tag{4}$$

where $k = 0, 1, 2, .., N$

We assume X has an asymptotic mean to$r(k)$:

$$r(k) \cong k^{-\beta} L_1(k) \tag{5}$$
$$k \to \infty, 0 < \beta < 1$$

The value of $L_1(k)$ gradually changes when its limit is taken to infinity:

$$\lim_{t\to\infty} \frac{L_1(tx)}{L_1(t)} = 1 \text{ for all } X > 0 \tag{6}$$

The value of L_1 at (t) is a constant, $L_1(t) = \log(t)$

In order to measure the self-similarity behavior, the Hurst parameter H is defined as follows:

$$H = 1 - \frac{\beta}{2} \tag{7}$$

Hurst parameter highly depends on parameter β of Equation (5). If the value of H is between 0.5 and 1, then it is considered a self-similar distribution [41].

$$0.5 < H < 1 \tag{8}$$

This behavior is known as long-range dependence where it is characterized by having a slowly decaying autocorrelation $r(k)$ (Equation 5), which is in contrast to the short-range dependence distribution of Poisson and Markov models. The long-range dependence process is achieved by relying on heavy-tailed distributions such as Pareto and Weibull that have a hyperbolic shape. On the other hand, the short-range dependence is described by light-tailed distributions such as exponential that decay exponentially over the distribution.

Among the well-known traffic generators, ReaSE and TrafGen both use exponential distribution in order to generate network traffic. ReaSE combines both mentioned mechanisms (multiple traffic sources and packet-level modification) and adopts a reasonable mixture of different protocols based on TCP, UDP, and ICMP to create eight different traffic profiles and assign a selection probability to each one of these profiles. On the other hand, TrafGen focuses on the parametric configuration of the hosts such

Table III. Traffic sources with different flow percentages.

Traffic source	Protocol	Flow (%)
HTTP traffic	TCP	60%
FTP traffic	TCP	10%
Telnet traffic	TCP	10%
Echo traffic	TCP	10%
UdpBurst traffic	UDP	5%
Ping traffic	ICMP	5%

TCP, Transmission Control Protocol; UDP, User Datagram Protocol; ICMP, Internet Control Message Protocol.

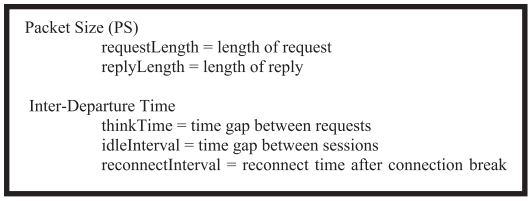


Figure 7. Main parameters in traffic generation.

as IDT, packet size, on length, and off length to generate a self-similar traffic pattern.

In OPNET [42], which is a commercial version of OMNeT++, there are built-in functions that produce Pareto and Weibull distributions. These libraries are also within the OMNeT++, and we will be using Pareto to generate self-similar traffic. Our traffic generation module is inspired from both TrafGen and ReaSE, where the important parameters are extracted from each and integrated into our framework. We created in OMNeT++ traffic generation modules that include the parameters used in ReaSE and the parameters currently used in INET framework. We manually configured the hosts to incorporate the protocols presented in Table III. The table also shows the traffic sources along with the traffic flow percentage. Hence, this approach is different from that used in ReaSE, which adopts a random traffic selection approach.

In our implementation, the generated traffic consists of variable traffic pattern that is achieved by configuring the numeric parameters to random or fixed values in the initialization file (*.ini*). Moreover, we use different TCP and UDP applications such as Telnet, Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and UDP to make use of multiple traffic sources. For that, we use the recent INET framework that provides different TCP and UDP applications. One such application is TCPBasicClientApp, which produces HTTP and FTP traffic. The behavior of each application is defined by setting packet size and IDT of the parameters as shown in Figure 7.

Table V. Parametric analysis of Telnet traffic.

Parameters	Mean	Distribution
Number of commands	10	Exponential
Command length	10 bytes	Pareto
CommandOutput length	40 bytes	Pareto
Think time	2 s	Exponential
Idle interval	3 s	Lognormal, truncnormal
Reconnect interval	30 s	fixed

The numeric parameters have been taken from analyzing different traffics in real-life scenarios. Table IV presents the analysis of HTTP traffic behavior based on a study by [43]. The use of such parameters can be integrated into our simulator by specifying corresponding distribution functions such as lognormal, Weibull, and Pareto. For instance, we use Pareto (α, X) distribution in assigning the request size by making $\alpha = 1$ (shape parameter) and $X = 360$ B (Mean of the HTTP request size).

According to [45], the FTP numeric parameters such as think time, idle interval, and reconnect interval are the same as HTTP, while request size, reply length, and number of requests vary to 20 bytes, 1 000 000 bytes, and 3, respectively [46]. The numerical parameters corresponding to Telnet are different from those of HTTP and FTP because its traffic behavior is not similar. In [44], the analysis of Telnet traffic shows that the IDT is represented by an exponential distribution while the size of data follows a Pareto distribution. In Table V, we present the assigned numeric values to our parameters.

The other applications such as ping, Udpburst, and Echo traffic are taken from the built-in applications in the INET framework of OMNeT++.

Figure 8 demonstrates a sample traffic pattern generated by our traffic generation module. The traffic consists of TCP, UDP, and ICMP packets. The number of packets varies from a minimum of 50 to a maximum of 590 packets at a time. More than 420 000 packets were injected from a total of 90 hosts. Figure 8 represents the captured traffic at Precision Time Protocol (PTP) core router that con-

Table IV. Parametric analysis of Hypertext Transfer Protocol traffic.

Parameters	Mean	Median	SD	Distribution
Request size	360.4 bytes	344 bytes	106.52	Pareto
Reply length	2000 bytes [44]	—	—	Pareto
Think time	0.86 s	0.17 s	2.15	Exponential
Idle interval	39.45 s	11.71 s	92.57	Weibull
Reconnect interval	30 s [44]	—	—	Fixed
Number of requests	1.74	1	1.74	Lognormal, truncnormal

SD, standard deviation.

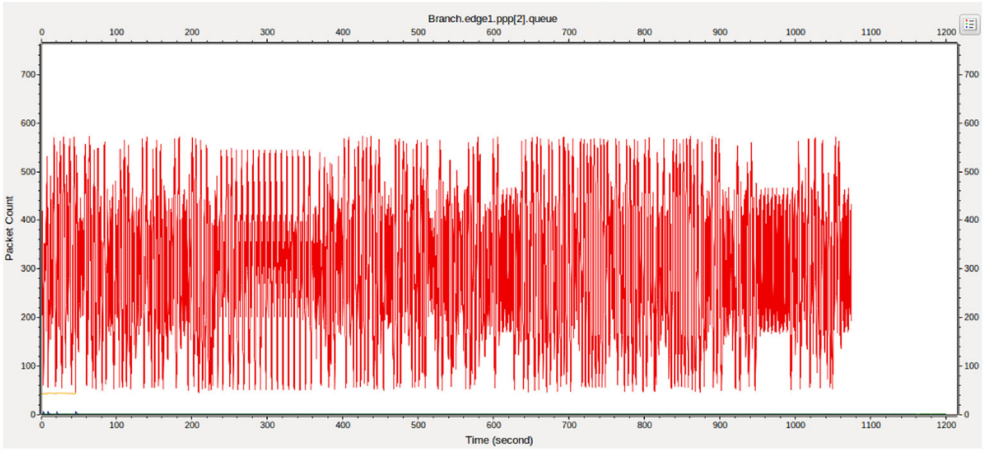


Figure 8. Traffic generation behavior using our module (packet count vs. time in seconds).

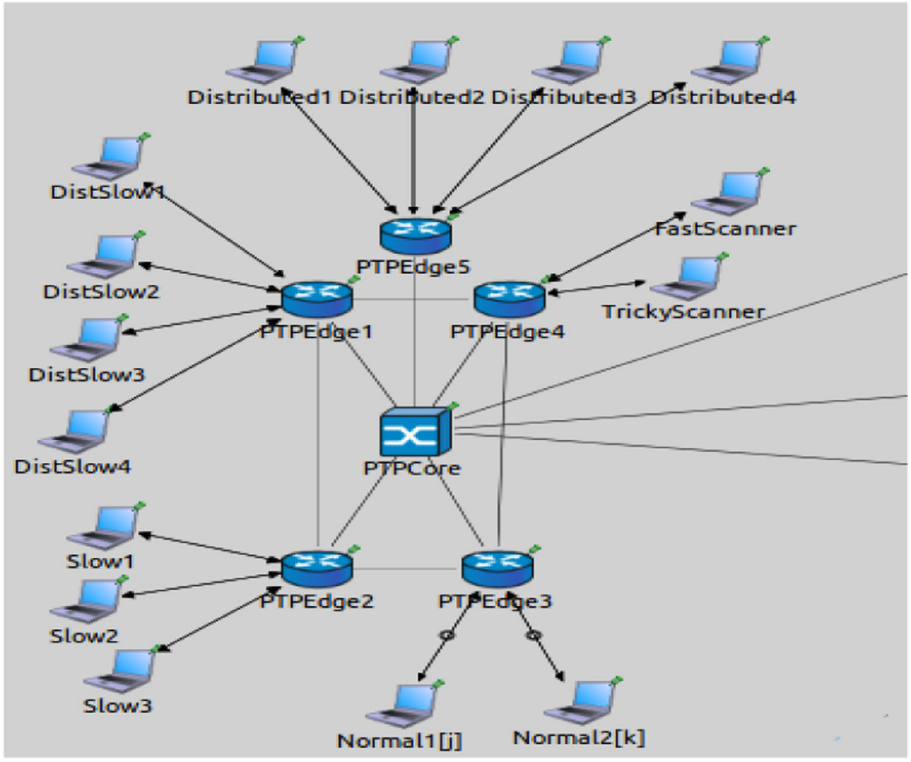


Figure 9. Peer-to-Peer Autonomous Systems including the malicious nodes.

nects the Peer-to-Peer AS to the whole network. This figure presents the variation in traffic behavior and contains many bursts in the timescale, which resembles self-similarity. These data are in accordance with the parameters presented in Table III.

In order to test the self-similarity of traffic and its realistic behavior, we analyzed the generated traffic in Wireshark [47], which is a known network protocol analyzer, and obtained compatible results compared with the data in Table III with 0.05% error in having malformed packets (118 packets). Moreover, we calculated the Hurst (H) [41] parameter of our traffic, which is used as a metric to show self-similarity in processes, and proved our result by ranging from 0.7 to 0.8.

3.3. Attack traffic

In our implementation of the attack traffic, we modified the INET tcp source files (TCPConnection.h, TCPConnectionBase.cc, TCPConnectionRcvSegment.cc, TCPConnectionUtil.cc, TCPConnectionEventProc.cc, and TCP.cc) to launch Connect, SYN, and FIN port scanning

attacks that are explained in Section 2. In our used simulation topology (Figure 9), we divide the attackers into three groups of normal scan, slow scan, and distributed-slow scan, which probe a range of 200 ports each.

(1) *Normal scan* It is the activity of sending consecutive messages to scan a range of ports. We divide the scanners into fastScanner and trickyScanner to produce different behaviors. The fast scanner simply launches the scan in a consecutive way, while the tricky scanner initiates eligible connections between the scans so that it goes undetected. One scenario of launching tricky scan is starting authorized connections after three consecutive scans and keep sending data until the next probe.
(2) *Slow scan* It is the kind of scanning where the intruder sends a probe to a certain port and waits for some time before sending the consecutive probe. This waiting time, chosen appropriately via testing, makes the traffic resemble normal behavior and thus deceiving most IDSs. We perform three different waiting timing approaches to test the detection of our IDSs. In Figure 9, *slow1* uses a 10-s time interval between each probe, while *slow2* and *slow3* launch their scans in 15-s and 30-s time intervals respectively.

    - Slow1 – 10 s time interval
    - Slow2 – 15 s time interval
    - Slow3 – 30 s time interval

    The selection of time interval highly depends on the number of port scans that is acquired by the attacker. Table VI estimates the amount of time elapsed in probing the given number of ports in hours with different time intervals. The number of ports is divided into well-known, registered, and dynamic ports that are 1024, 48 128, and 16 385 ports respectively.
(3) *Distributed scan* Distributed port scanning is the kind of scanning where the intruder launches a coordinated scan distributed among multiple parties. Each party selects a set of the available ports and scans them. The attacker then collects the scan result from every party. Such an attack has many similarities with distributed denial-of-service attacks that have long been a challenge to the research community. In our used topology (Figure 9), we have four hosts named Distributed1, 2, 3, and 4 that launch coordinated scan, which cover a range of 200 ports divided among them. We also devised a tricky distributed scan that initiates authorized connections between scanning activity to make this approach a more challenging scan.
(4) *Distributed and slow scan* Detecting distributed port scanning remains to be a difficult task, but when combined with slow port scanning, the problem becomes even more challenging. In our used topology, DistSlow1, 2, 3, and 4 (Figure 9) launch a coordinate slow scan with 5 s time interval between each scanner host that probes a range of 200 ports. For instance, DistSlow1 starts by scanning port 80, then after 5 s, DistSlow2 scans port 81 and so on.

    - DistSlow1 – 80, 84, 88, ...
    - DistSlow2 – 81, 85, 89, ...
    - DistSlow3 – 82, 86, 90, ...
    - DistSlow4 – 83, 87, 91, ...

    By devising distributed-slow scan, the attackers can probe larger scale of ports in a faster amount of time compared with only slow scanning. Table VII gives an estimate of probing time in hours with four scanning hosts in different time intervals.

Table VI. Slow scanning time estimate for probing ports in hours.

Time interval (s)	Well-known 1024 (h)	Registered 48 128 (h)	Dynamic 16 385 (h)	All 65 536 (h)
5	1.42	66.84	22.76	91.02
10	2.84	133.69	45.51	182.04
15	4.27	200.53	68.27	273.07
20	5.69	267.38	91.03	364.09
30	8.53	401.07	136.54	1546.13
60	17.07	802.13	273.08	1092.27

Table VII. Distributed and slow scanning time estimate for probing ports in hours with four scanning hosts.

Time interval (s)	Well-known 1024 (h)	Registered 48 128 (h)	Dynamic 16 385 (h)	All 65 536 (h)
5	0.36	16.71	5.69	22.76
10	0.71	33.42	11.38	45.51
15	1.07	50.13	17.07	68.27
20	1.42	66.84	22.76	91.02
30	2.13	100.27	34.14	136.53
60	4.27	200.53	68.27	273.07

4. TESTING RESULTS

To create the benchmark, we launch the mentioned attacks in Section 3 along with normal traffic generated from Peer-to-Peer AS (Normal1[j] and Normal2[k], where j and k vary from low to high depending on the load of traffic) to three different network environments that are Cloud AS, University AS, and Private Enterprise AS. The same types of attacks are launched to all the ASs with different background activity that vary from low and medium to high load of traffics in order to test the detection approach and robust performance of the IDSs.

The IDS should protect the network from inbound and outbound connections to make sure no adversary bypasses

Table VIII. Comparison between snort and MalwareAnalysis in scan detection.

Scan type	Snort	MalwareAnalysis
Slow1 (5 s)	Detected	Detected
Slow2 (10 s)	Not detected	Detected
Slow3 (15 s)	Not detected	Not detected
Slow4 (30 s)	Not detected	Not detected
Distributed	Detected as distributed	Detected as scan
Distributed tricky	—	—
DistSlow1 (5 s)	Not detected	Detected
DistSlow2 (5 s) + tricky	Not detected	Not detected
DistSlow3 (10 s)	Not detected	Detected
DistSlow4 (10 s) + tricky	Not detected	Not detected

its security protocol. Keeping this measurement in mind, we kept the simulation running for 20 min while recording the log on different routers such as UnEdge1&2, CloudCore, and PEgw1&2 because they connect the servers to internal and external hosts and can be visualized as the IDS ideal position. Moreover, we analyze the recorded log in the PCAP format using MalwareAnalysis [48] (PCAP and traffic analyzer) and Snort [5] (a free IDS) under Ubuntu 12.04.

In Table VIII, we summarize the comparison of Snort and MalwareAnalysis in detecting different port scanning activity. MalwareAnalysis and Snort were able to successfully detect all the port scanning attacks launched from normal scanners; however, Snort was not able to detect any of the slow scans. In MalwareAnalysis, the slow scan performed by Slow1 (10 s interval) was easily detected, while the other two slow scanners Slow2 and 3 went unnoticed. The distributed scans were detected in Snort as distributed approach, while in MalwareAnalysis, it was just considered as a scan.

With the Distributed tricky approach in which we initiate eligible connection between scans, the detections were variable based on number of scans between each eligible connection. In distributed and slow scanning, the results showed that MalwareAnalysis raised an alarm due to its flow-based [3] approach in detecting anomalous activities, which relies on checking the log within a specified time window and not on tracking traces of packets. This way, when there are many port scans within its time window, then they can be detected easily. To tackle this detection approach, we also devise a tricky version of distributed and slow to let our scans go unnoticed.

In Table IX, we show the robust performance of Snort and MalwareAnalysis with different loads of traffic in the background activity. These loads vary from low and medium to high loads of traffic. Our results show that Snort analyzes the traffic much faster than MalwareAnalysis. However, in terms of accuracy, MalwareAnalysis detects more port scans than Snort. Consequently, our log is capable of testing the robust performance, which is an important metric in evaluating IDSs along with the hit rate and false alarm ratios.

Table IX. Robust performance of Snort and MalwareAnalysis with different loads of traffic.

Load of traffic	Snort	MalwareAnalysis
	Time (s)	Time (s)
Low (20 MB)	6	20
Medium (70 MB)	5	100
High (140 MB)	45	230

5. CONCLUSION

In this work, we presented a simulation framework that we used to create realistic traffic logs with entries annotated as malicious or not. Our major aim was to create network logs that resemble as much as possible real-life traffic. To do that, we created realistic modules for the network topology, good traffic, and bad traffic. Different types of port scans such as slow, distributed, and distributed-slow scans were implemented and injected within the normal traffic. OMNeT++ was used for simulations. We tested two IDSs, specifically Snort and MalwareAnalysis, under different background activities with low to high loads of traffic and tested their robust performance. Our results show that most of the attacks were not detected by the IDSs, which makes our benchmarks a great tool for testing different algorithms. The proposed testing approaches can tackle the hurdles of testing IDSs by comparing the results of the detection experiment with prior knowledge of number of launched attacks. Consequently, the hit rate and false alarms ratio can be tested along with the robust performance of the IDS. Our future work includes further testing the developed port scanning benchmarks with commercial IDSs.

ACKNOWLEDGEMENTS

This work was partially funded by the American University of Beirut Research Board (AUB-URB) and an American University of Sharjah (AUS) Research Grant.

REFERENCES

1. Gamer T, Mayer CP. Large-scale evaluation of distributed attack detection. In *Proceedings of the 2nd International Workshop on OMNeT*, Brussels, Belgium, 2009; Article No. 68.
2. Barry BI. Intrusion detection with OMNeT. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Rome, Italy, 2009; Article No. 5.
3. Bhuyan MH, Bhattacharyya D, Kalita J. Surveying port scans and their detection methodologies. *The Computer Journal* 2011; **54**(10): 1565–1581.

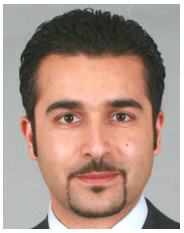
4. Hacker Watch, *Anti-hacker Community*. Available at: http://www.hackerwatch.org/ [Accessed on 01 March 2013].
5. *SNORT*. Available at : http://www.snort.org [Accessed on 01 March 2013].
6. Dokas P, Ertoz L, Kumar V, Lazarevic A, Srivastava J, Tan P. Data mining for network intrusion detection. In *Proceedings of the NSF Workshop on Next Generation Data Mining*, Baltimore, MD, 2002; 21–30.
7. Soniya B, Wiscy M. Detection of TCP SYN scanning using packet counts and neural network. In *Proceedings of the IEEE International Conference on Signal Image Technology and Internet Based Systems (SITIS)*, Bali, Indonesia, 2008; 646–649.
8. Verwoerd T, Hunt R. Intrusion detection techniques and approaches. *Computer Communications* 2002; **25** (16): 1356–1365.
9. Lippmann RP, Fried DJ, Graf I. *et al.* Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition*, 2000; 12–26.
10. The UCI KDD Archive. Information and Computer Science University of California, Irvine. Available at: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [accessed on 01 March 2013].
11. Wan T, Yang XD. IntruDetector: a software platform for testing network intrusion detection algorithms. In *Proceedings of the Computer Security Applications Conference (ACSAC)*, New Orleans, Louisiana, USA, 2001; 3–11.
12. Zhou S, Zhang G, Zhang G, Zhuge Z. Towards a precise and complete internet topology generator. In *Proceedings of the International Conference on Communications, Circuits and Systems*, Guilin, Guangxi, China, 2006; 1830–1834.
13. Dietrich I. OMNeT++ Traffic Generator, Sept. 2006. Available at: http://www7.informatik.unierlangen.de/∼isabel/omnet/modules/TrafGen/.
14. Giruka V, Singhal M, Royalty J, Varanas S. Security in wireless sensor networks. *Wireless Communications and Mobile Computing* 2008; **8**(1): 1–24.
15. Fragkiadakis A, Siris V, Petroulakis N, Traganitis A. Anomaly based intrusion detection of jamming attacks, local versus collaborative detection. *Wireless Communications and Mobile Computing* 2013. DOI: 10.1002/wcm.2341.
16. Hai TH, Huh EN, Jo M. A lightweight intrusion detection framework for wireless sensor networks. *Wireless Communications and Mobile Computing* 2010; **10**(2): 559–572.
17. Gamer T. Collaborative anomaly-based detection of large-scale internet attacks. *Computer Networks* 2012; **56**(1): 169–185.
18. The NSS Group 2003. Intrusion Detection System Group Test (Edition 4). Available at: http://www.nss.co.uk [Accessed on 01 March 2013].
19. National Laboratory for Applied Network Research, 2003. *Network Traffic Packet Header Traces*. Available at: http://pma.nlanr.net [Accessed on 01 March 2013].
20. Roemer B. BonnTraffic: a modular framework for generating synthetic traffic for network simulations, Nov. 2005. Available at: http://web.informatik.uni-bonn.de/IV/bomonet/BonnTraffic.htm [Accessed on 01 March 2013].
21. http://www.ietf.org/rfc.html [Accessed on 01 March 2013].
22. Pang R, Allman M, Bennett M, Lee J, Paxson V, Tierney B. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, Berkeley, CA, USA, 2005; 2–2.
23. Kim SK, Lee SH, Seo SW. An automatic portscan detection system with adaptive threshold setting. *Journal of Communications and Networks* 2010; **12** (1): 74–85.
24. Sommers J, Yegneswaran V, Barford P. Toward comprehensive traffic generation for online ids evaluation. *Technical Report* , University of Wisconsin, 2005.
25. Gates C. Coordinated scan detection. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2009; 11–20.
26. White B, Lepreau J, Stoller L. *et al.* An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review* 2002; **36**: 255–270.
27. (Anthraxx) DR, (Kolrabi) BP. DScan Software. 2002. Available at: http://www.u-n-f.com/dscan.html [Accessed on 01 March 2013].
28. Mixter. Network security analysis tools. Available at: http://nsat.sourceforge.net/ [Accessed on 01 March 2013].
29. Alon N, Moshkovitz D, Safra S. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)* 2006; **2**(2): 153–177.
30. Gamer T, Scharf M. Realistic simulation environments for IP-based networks. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Marseille, France, 2008; Article No. 83.
31. Mayer CP, Gamer T. Integrating real world applications into OMNeT. *Technical Report TM-2008-2*, Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, 2008.
32. University of Oregon. Route Views Project. Available at: http://www.routeviews.org [Accessed on 01 March 2013].
33. Quoitin B, Van den Schrieck V, François P, Bonaventure O. IGen: generation of router-level inter-

net topologies through network design heuristics. In *Teletraffic Congress*, Paris, France, 2009; 1–8.
34. Medina A, et al. Brite. Available at: http://www.cs.bu.edu/brite [Accessed on 01 March 2013].
35. Benson T, Akella A, Maltz DA. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, Melbourne, Australia, 2010; 267–280.
36. NTT Group. Available at: http://www.nttdata.com/global/en/ [Accessed on 01 March 2013].
37. Avallone S, Emma D, Pescap A, Ventre G. A practical demonstration of network traffic generation. In *Proceedings of the Internet and Multimedia Systems and Applications (IMSA)*, Kauai, Hawaii, USA, 2004; 138–143.
38. Willinger W, Taqqu MS, Sherman R, Wilson DV. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE Transactions on Networking* 1997; **5**(1): 71–86.
39. Sager P. Does circuit emulation in metropolitan gigabit ethernets require service priority. *Post Diploma Thesis NA-2005-02*, Swiss Federal Institute of Technology Zurich, 2005.
40. Park K, Willinger W. *Self-similar Network Traffic and Performance Evaluation*. Wiley Online Library: Malden, MA, USA, 2000.
41. Fras M, Mohorko J. Estimating the parameters of measured self similar traffic for modeling in OPNET. In *Proceedings of the International Workshop on Systems, Signals and Image Processing*, Maribor, Slovenia, June 2007; 78–81.
42. OPNET. Available at: http://www.opnet.com/ [accessed on 01 March 2013].
43. Choi H, Limb JO. A behavioral model of web traffic. In *Proceedings of the International Conference on Network Protocols (ICNP)*, Toronto, Canada, 1999; 327–334.
44. Wireshark. Available at: http://www.wireshark.org/ [Accessed on 01 March 2013].
45. OMNeT++. INET Framework. http://inet.omnetpp.org/ [Accessed on 01 March 2013].
46. Yuksel M. Traffic generator for an on-line simulator. *Master's Thesis*, Department of Computer Science, Rensselaer Polytechnic Institute, 1999.
47. Dabbagh M, Ghandour AJ, Fawaz K, Hajj W, Hajj H. Slow port scanning detection. In *Proceedings of the International Conference on Information Assurance and Security (IAS)*, Malacca, Malaysia, 2011; 228–233.
48. Botterill D. PCAP Analyzer. Available at: http://www.cs.bham.ac.uk/~tpc/PCAP/ [Accessed on 01 March 2013].

AUTHORS' BIOGRAPHIES

Wassim El-Hajj is Assistant Professor and Chair of the Computer Science department at the American University of Beirut (AUB). He received his BS degree from AUB in 2000 and the MS and PhD degrees in 2002 and 2006, respectively, from Western Michigan University (WMU), all in Computer Science. His research interests are in the areas of wireless communication, network security, and data mining. His research activities culminated with more than 50 publications, published in reputable journals such as IEEE Transaction on Vehicular Technology, IEEE Transactions on VLSI, and IEEE Sensors, as well as top IEEE conferences that include ICDM, ICC, AINA, GLOBECOM, WCNC, and ISWTA. He is the recipient of numerous recognitions, most notably, the WMU Excellence in Research Award for three years in a row.

Mustafa Al-Tamimi received his BSc and MSc degrees in Computer Science from the American University of Beirut, Lebanon, in 2009 and 2014, respectively. At present, he is working as an IT security officer in a pharmaceutical company (Broadmed). He was also a graduate assistant at the American University of Beirut from 2010 to 2012. Most recently, he has focused more on research on the behaviors of intrusion detection systems, with special attention to realistic simulations.



Fadi Aloul (M'03-SM'08) is currently an Associate Professor of Computer Science and Engineering at the American University of Sharjah, UAE. Dr. Aloul received his MS and PhD degrees in Computer Science and Engineering from the University of Michigan, Ann Arbor, USA, and a BS degree in Electrical Engineering summa cum laude from Lawrence Technological University, Michigan, USA. He was a post-doc research fellow at the University of Michigan during summer 2003 and a visiting researcher with the Advanced Technology Group at Synopsys during summer 2005. He is a Certified Information Systems Security Professional (CISSP). Dr. Aloul received a number of awards including the prestigious Sheikh Khalifa, UAE's President, Award for Higher Education, AUS Excellence in Teaching Award, Abdul Hameed Shoman Award for Young Arab Researchers and the Semiconductor Research Corporation (SRC) Research Fellowship. He has 98 publications in international journals and conferences, in addition to one US patent. His current research interests are in the areas of design automation, optimization, and computer security.