Framework for Creating Realistic Port Scanning Benchmarks

Mustafa Al-Tamimi
Department of Computer Science
American University of Beirut
Beirut, Lebanon
mai26@aub.edu.lb

Wassim El-Hajj
Department of Computer Science
American University of Beirut
Beirut, Lebanon
we07@aub.edu.lb

Fadi Aloul
Department of Computer Engineering
American University of Sharjah
Sharjah, UAE
faloul@aus.edu

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. However, very little work has been done on generating port scanning benchmarks that researchers can use to test their detection methods. In this paper, 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 (topology creation, good traffic generation, bad traffic generation), each of which we make realistic, similar to deployed and usable networks. Hence the resultant benchmark is annotated and made public.

Keywords— Port Scanning Benchmarks, Port Scanning, Intrusion Detection System.

I. 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 to the cost and efforts spent in establishing a real test bed 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.

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. Port scanning 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 [4]. 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. Due to their dangerous consequences, 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 corporate 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. These tests can be either performed in real environments or simulated ones [9]. Currently, researchers perform these tests under simulated environments, because of the high costs and risks involved in testing under real environments [10]. Meanwhile, successful and effective evaluation of IDSs requires overcoming several challenges summarized hereafter. A challenging task 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 [10]. Another challenging task that is crucial in IDS testing is the generation of background traffic. Most IDS testing approaches can be classified into one of four environments depending on the background traffic generation patterns:

1. No back ground 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 [9].
2. Real background traffic: Testing in real environment with realistic background traffic is very effective in

978-1-4673-2480-9/13/$31.00 ©2013 IEEE

finding the hit rate and false alarm ratio because of the background activity. However, it is impossible to guarantee the identification of all attacks since there is no prior knowledge about hidden ones. Another major drawback of such approach is that the data cannot be distributed due to privacy concerns [11].

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 [12].
4. Generated background traffic: In this approach the background traffic is generated by complex traffic generators that model realistic traffic behavior [13]. Since it is guaranteed that the generated traffic doesn't 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 paper, we adopt the last environment and develop a comprehensive framework that generates benchmarks for port scanning testing. 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++. 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 does not contain many port scanning activities or contains manually generated port scanning traffic [4]. The rest of this paper is structured as follows: Section II summarizes previous works on port scanning benchmarks development. Section III briefly overviews OMNet++. Section IV describes the topology, background, and attack traffic generation. Section V concludes the paper and presents our future work.

II. RELATED WORK

In recent studies, researchers have proposed several ways to test IDSs under various environments (real background traffic, no back ground traffic, or generated background traffic). For real background traffic, different data sets have been used for testing purposes. The authors in [14] publicly released internal enterprise traffic, known as Lawrence Berkeley National Laboratory data sets (LBNL), that spans more than 100 hours 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 [4], a data set known as Endpoint background traffic was presented. The data set consists of data exchanged in home and university environments. Since 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 [15], the authors gathered real traffic traces from a dormitory at Seoul National University (SNU) to test their proposed port scanning detection method. They used a fast increase slow decrease (FISD) scheme that automatically and adaptively sets the port scanning detection threshold, based on traffic statistical data during prior time periods. The proposed method outperformed Snort, BRO, and Threshold Random Walk (TRW).

In [16], the authors created a port scanning testbed using Deter network that uses Emulab software [18]. DScan [19] and NSAT [20] 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 [17]. 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 VoIP IDS. However, the generated traffic didn't convey a realistic behavior to test the false alarm ratio.

In [21], the authors built a model called Realistic Simulation Environment (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 realistic simulation environment 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 [22]. 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. We next give an overview of our simulator of choice: OMNeT++.

III. OMNeT++ Highlights

OMNeT++ is a discrete event simulator which 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 networks and each network contains simple and/or compound modules where each compound module consists of multiple simple modules as shown in Figure 1.

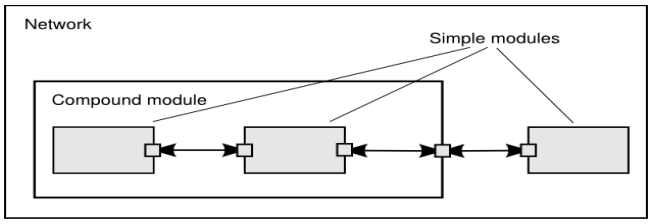

Figure 1. OMNeT++ Model Structure

The main components of each simulation are the NED and INI files. The NED which is the network description file (*.ned*), describes the structure and parameters of nodes in a network. The initialization file (*.ini*) is used to set values to the parameters included in the NED, for instance 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 Transmission Control Protocol (TCP), Internet Protocol (IP), User Datagram Protocol (UDP), 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 (RFC) document [23].

IV. 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.

A. Network Topology

To generate realistic topologies, two approaches are mainly used [21]. The first approach is based on observing real-life scenarios collected from BGP routing data or Routeviews project [24]. The main drawback of real observations is that the collected data is not easily integrated with the simulator given the heavy load of data. Moreover, the observed data are not updated based on current topologies [21]. The second approach relies on random topology generation. However, random networks do not accurately model real topologies since major parameters such as link metrics and internal BGP configurations are often ignored [25]. Even with this drawback, the research community heavily uses random topology generation that depends on power-low distribution [25]. In this 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 doesn't take into account the realistic parametric values in link speeds and routers. Another option for topology generation is BRITE [26], which can be integrated with NS-2 and OMNeT++.

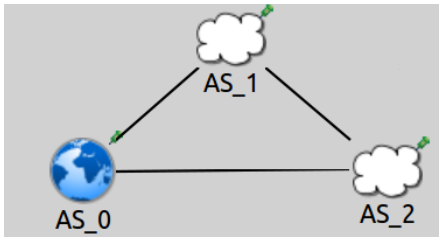

Figure 2. Network topology combining all the Autonomous Systems.

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. In ReaSE, the generation of realistic topologies is divided into two parts due to the hierarchical structure of the Internet. On one hand, Autonomous Systems (AS) level topology focuses on the connection of multiple separate domains as shown in Figure 2. On the other hand, the router level topology of each AS has to be generated. This method is based on Positive-Feedback Preference (PFP) that randomly implements power-low distribution to the nodes [27].

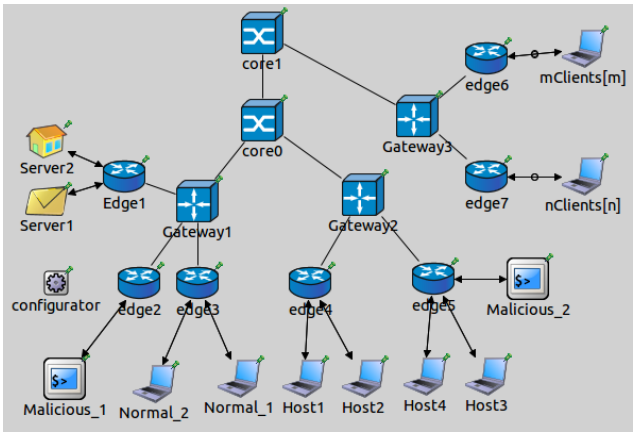

Figure 3. Router Level Topology (core, gateway, edge)

The core routers (*core0* and *core1* in Figure 3 for example) are all connected to each other through very

high speed links, especially since they are used to connect together the various Autonomous Systems. Furthermore, each core connects to few gateway routers via high speed links (for instance in Figure 3, *core0* connects to *Gateway1* & *Gateway2*) and each gateway is connected to multiple edge routers. Finally, the edges connect to several hosts with lower speed links. Figure 3 represents the proposed router level topology and Table I presents the chosen parameters. The second row of Table I (*Core<----2.5Gbps--->Core*) can be read as follows: bi-directional links with speed 2.5 Gbps connect Core routers. The last row means that Hosts connect to Edge Routers via uni-directional links with speed 0.128 Mbps. The other rows follow the same explanation.

Table I. Links properties in the proposed topology

Router Level	Link Speed	Router Level
Core	<---- 2.5 Gbps --->	Core
Core	<----- 1 Gbps ---->	Gateway
Gateway	<--- 155 Mbps--->	Edge
Edge	<---- 10 Mbps---->	Server
Edge	-- 0.768 Mbps--->	Host
Host	-- 0.128 Mbps--->	Edge

B. *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 require the generation of self-similar behavior [16] 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 [21], BonnTraffic [13], TrafGen [28], and D-ITG [29]. One possibility to achieve self-similar traffic behavior is to use multiple traffic sources that are switched on and off based on heavy-tailed intervals [30]. Another possibility is to produce traffic at packet level by replicating appropriate stochastic processes for both Inter Departure Time (IDT) and Packet Size (PS) random variables (exponential, uniform, Cauchy, normal, pareto, etc.) [29].

Hence, 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, TraffGen (mentioned above) focuses on the parametric configuration of the hosts such as inter departure time, packet size, ON length, and OFF length to generate a self-similar traffic pattern.

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 (old version) and the parameters currently used in INET (the new ReaSE). We manually configured the hosts to incorporate the protocols presented in Table II. 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.

Table II. Traffic sources with different flow percentages

Traffic Source	Protocol	Flow (%)
Http traffic	TCP	32 %
Ftp traffic	TCP	20 %
Telnet traffic	TCP	10 %
Echo traffic	TCP	33 %
UdpBurst traffic	UDP	2 %
Ping traffic	ICMP	3 %

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, HTTP, 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 that produces HTTP and FTP traffic by setting the parameters shown in Figure 4.

```
FTP:
   numRequestsPerSession = exponential(3)
   requestLength = truncnormal(20,5)
   replyLength = exponential(1000000)
HTTP:
   numRequestsPerSession = 1
   RequestPerSession = exponential(5)
   requestLength = truncnormal(350,20)
   replyLength = exponential(2000)
```

Figure 4. FTP and HTTP parameters

Our use of truncnormal and exponential functions varies the parameter's values, and hence results into a variable traffic behavior. Figure 5 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 thousands packets were injected from a total of 90 hosts (note that, in Figure 3 *mClients* and *nClients* are arrays of *30* and *40* hosts respectively). Figure 5 represents the captured traffic at *Edge1* router that connects Servers 1 & 2 to the whole network. This figure assures that the traffic behavior of our generator is self-similar and realistic in accordance with the parameters presented in Table II.

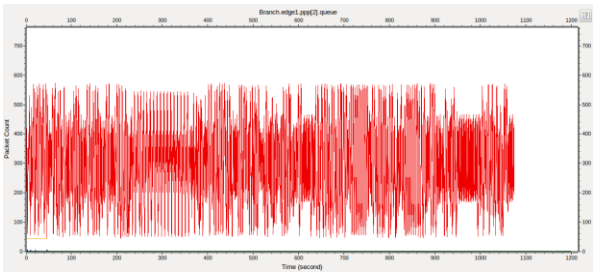

Figure 5. Traffic generation behavior using our module

C. Attack Traffic

In order to generate the attack traffic, first we take an insight into the port scanning activity. Basically, there are 65536 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) [4]. 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 occur using UDP which 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 due to the connectionless services [4].

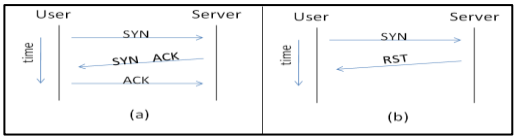

Figure 6. TCP 3-way handshake. (a) The 3-way handshake with an open port. (b) Connection attempt on a closed port.

A TCP connection is established by a 3-way handshake which is explained in Figure 6, 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 a SYN flag. If the port is open on the server side, it will respond with a TCP packet containing the SYN+ACK flag after which the initiating user will respond with a TCP ACK message and finally the connection is established. On the other hand, if the port is closed the server will reply with a TCP containing RST flag [31].

Among the well-known port scanning attacks that use TCP, the following are the mostly used [4]: **Connect Scan:** a TCP connect scan completes the 3-way handshake and after successful attempt it is logged as a connection. If the connection is successful, the attacker sends a FIN packet to tear down the connection. This type of scan is recorded in the log. **SYN Scan:** It is considered the most popular type of port scanning and usually referred to 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 not completing the 3-way handshake. This way the scan doesn't show up in the application level logs since the connection is not established. This way gives more advantage to the scanner to remain stealthy. **FIN Scan:** This type of attack is used when the network firewall drops all SYN-ACK packets to 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.

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. For testing purposes we modified the scanning process in such a way that after targeting a specific port number, we target the 4 ports just after it.

In our used simulation topology (Figure 3), Server1 has 5 open ports (80, 200, 300, 1000, and 2000), while Server2 has only 2 open ports (80 and 1000). We set Malicious_1&2 and nClients[0..29] to launch the attacks by targeting the ports and incrementing them up to 4. For instance Malicious_1 (row 2 of Table III) launches a FIN attack from its local port 22 to port 1000 on Server 1, then 1001, 1002, 1003, and 1004. Similarly, Malicious_1 launches another FIN attack from port 23 to port 200 on Server 1, followed by scanning ports 201, 202, 203, and 204. On the other hand and to attempt to fool IDSs, we set each of the mClients[30] hosts to generate normal HTTP and FTP traffic to Server 2 resulting in 60 different connections. Table III shows the simulated attack traffic.

Table III. Attack traffic statistics

Source Name	Source Port	Port Scan	Dest Name	Dest Port
Malicious_1	22	FIN	Server 1	1000
Malicious_1	23	FIN	Server 1	200
Malicious_1	26	SYN	Server 1	80
Malicious_1	27	SYN	Server 1	300
Malicious_2	18	FIN	Server 1	2000
Malicious_2	24	FIN	Server 1	1000
Malicious_2	33	SYN	Server 1	80
Malicious_2	48	SYN	Server 1	3000
nClients[0..10]	19	FIN	Server 1	80
nClients[11..20]	20	FIN	Server 1	200
nClients[21..29]	21	FIN	Server 1	2000

To create the benchmark, we kept the simulation running for 20 minutes with a high load of traffic while recording the log on the Edge-1 router, since it connects the servers to other hosts and can be visualized as the IDS proper position. Then we analyze the recorded log in the PCAP format using MalwareAnalysis (PCAP analyzer that uses Snort database) under Ubuntu 12.04. MalwareAnalysis was able to successfully detect all the port scanning attacks we launched, but it also reported a lot of normal traffic as attacks (false alarms). Table IV shows a summary of the detected attacks. Row 2 of Table IV indicates that port 80 on Server 1 experienced multiple port scanning activities, out of which 63% are Maimon (an alternative for FIN scan) and 37% are connect scan. This is consistent with the attacks we generated (rows 4, 8, and 10 of Table III). However, we didn't attempt any port scan on Server 2, but MalwareAnalysis showed that there exist alarms on ports 80 and 1000 (last 2 rows of

Table IV). These alarms are false alarms and should not have been classified as attacks. This might be due to the connections established by mClients[30]. Hence, this shows the inability of MalwareAnalysis (which uses Snort) to correctly and accurately classify port scanning attacks. It also shows the usefulness of our generated traffic in analyzing port scanning detection systems. The code and port scanning benchmarks are made public on the following link:

http://staff.aub.edu.lb/~we07/Tools/PortScanningBenchmarks/

Table IV. Detected attacks using MalwareAnalysis

Destination : Port	Source	Description
Server 1 : 80	Multiple	Maimon:63% Connect: 37%
Server 1 : 1000	Multiple	Maimon:9% Connect:2% Other:89%
Server 1 : 300	Multiple	Connect: 100%
Server 1 : 200	Multiple	Maimon: 100%
Server 1 : 2000	Multiple	Maimon: 100%
Server 2 : 80	Multiple	Connect: 40% Other: 60%
Server 2 : 1000	Multiple	Other: 100%

V. Conclusion & Future Work

In this paper, 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. Two types of port scans were implemented and injected within the normal traffic. OMNeT++ was used for simulations. For future work, we plan to improve our prototype to become a complete environment that can generate multiple types of port scans as well as distributed and slow port scans.

References

[1] T. Gamer and C. P. Mayer, "Large-scale evaluation of distributed attack detection," in *Proc. of the 2nd Int'l Workshop on OMNeT (Hosted by SIMUTools 2009),* 2009.

[2] B. I. A. Barry, "Intrusion detection with OMNeT," in *Proc. of the 2nd International Conference on Simulation Tools and Techniques*, 2009.

[3] Hacker Watch, Anti-Hacker Community. *http://www.hackerwatch.org/ last visited: 12 December 2012.*

[4] Bhuyan, Monowar H., D. K. Bhattacharyya, and J. K. Kalita. "Surveying port scans and their detection methodologies." *The Computer Journal*, 54(10), 1565-1581, 2011.

[5] http://www.snort.org.

[6] P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava and P. Tan, "Data mining for network intrusion detection", In Proc. 2002

[7] B. Soniya and M. Wiscy, "Detection of TCP SYN Scanning Using Packet Counts and Neural Network," Signal Image Technology and Internet Based Systems, 2008. SITIS '08. IEEE International Conference, pp.646-649, Nov. 30 2008-Dec. 3 2008 doi: 10.1109/SITIS.2008.33

[8] Verwoerd, TH. and Hunt, R. Intrusion Detection Techniques and Approaches. The Computer Communications, 25, 1356-1365, 2001.

[9] T. Gamer, "Collaborative anomaly-based detection of large-scale internet attacks," Computer Networks, vol. 56, pp. 169, 2012.

[10] Wan, T. and Yang, X. 2001 IntruDetector: A Software Platform for Testing Network Intrusion Detection Systems. In Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001) (New Orleans, Louisiana, December 2001).

[11] The NSS Group 2003. Intrusion Detection System Group Test (Edition 4). Available: http://www.nss.co.uk.

[12] National Laboratory for Applied Network Research 2003.NLAR Network Traffic Packet Header Traces. Available: http://pma.nlanr.net.

[13] B. Roemer. BonnTraffic: A modular framework for generating synthetic traffic for network simulations, Nov. 2005. http://web.informatik.uni-bonn.de/IV/bomonet/BonnTraffic.htm.

[14] Pang, R.,Allman, M., Bennett, M., Lee, J., Paxson,V. andTierney, B. (2005)A First Look at Modern Enterprise Traffic. Proc. ACM IMC'05, Berkeley, CA, USA, October, 19–21, pp. 2–2. USENIX Association

[15] S. K. Kim, S. H. Lee and S. W. Seo, "An Automatic Portscan Detection System with Adaptive Threshold Setting," JOURNAL OF COMMUNICATIONS AND NETWORKS, vol. 12, 2010.

[16] C. Gates, "Coordinated scan detection," in Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 09), 2009, .

[17] N. Alon, D. Moshkovitz and S. Safra, "Algorithmic construction of sets for k-restrictions," ACM Transactions on Algorithms (TALG), vol. 2, 153-177, 2006.

[18] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. "An integrated experimental environment for distributed systems and networks." In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, 255 – 270, Boston, MA, USA, December 2002. USENIX Association.

[19] D. R. (Anthraxx) and B. P. (Kolrabi). DScan Software. http://www.u-n-f.com/dscan.html, 2002. Last visited: 12 June 2008.

[20] Mixter. Network security analysis tools. http://nsat.sourceforge.net/.

[21] T. Gamer and M. Scharf, "Realistic simulation environments for IP-based networks," in Proc. of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, 2008.

[22] C. P. Mayer and T. Gamer, "Integrating real world applications into OMNeT," Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech.Rep.TM-2008-2, 2008.

[23] http://www.ietf.org/rfc.html

[24] University of Oregon. Route Views Project. http://www.routeviews.org.

[25] B. Quoitin, V. Van den Schrieck, P. François and O. Bonaventure, "IGen: Generation of router-level internet topologies through network design heuristics," in Teletraffic Congress, 2009. ITC 21 2009. 21st International, 2009, pp. 1-8.

[26] http://www.cs.bu.edu/brite

[27] S. Zhoua, G. Zhang, G. Zhang, and Z. Zhuge. Towards a Precise and Complete Internet Topology Generator. In Proc. of ICCCAS, volume 3, pages 1830{1834, June 2006.

[28] I. Dietrich. OMNeT++ Traffic Generator, Sept. 2006. http://www7.informatik.uni-erlangen.de/~isabel/omnet/modules/TrafGen/.

[29] S. Avallone, D. Emma, A. Pescap, and G. Ventre. A Practical Demonstration of Network Traffic Generation. In Proc. of the 8th IMSA, pages 138-143, Aug. 2004.

[30] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level. In Proc. of ACM SIGCOMM, pages 100-113, Sept. 1995.

[31] M. Dabbagh, A. Ghandour, K. Fawaz, W. El Hajj, and H. Hajj "Slow port scanning detection." in Proc. of the 7th IEEE Intl Conference on Information Assurance and Security. 2011.