Big Data Testbed for Network Attack Detection

Dániel Csubák, Katalin Szücs, Péter Vörös, Attila Kiss

Department of Information Systems, Eötvös Loránd University
Pázmány Péter sétány 1/C, H-1117 Budapest, Hungary
csuby@caesar.elte.hu, szucsk@caesar.elte.hu, vopraai@inf.elte.hu,
kiss@inf.elte.hu

Abstract: Establishing an effective defense strategy in IT security is essential on one hand, but very challenging on the other hand. According to the 2014 Cyberthreat Defence Report [1] that involved more than 750 security decision makers and practitioners, more than 60% of organizations had been breached in 2013. Big data analytics in security provides the possibility to gather and analyse massive amounts of digital information in order to predict and prevent these attacks. However, since collecting the needed data in an efficient, complete and reliable fashion encounters problems, the industry is lacking and could truly benefit from a tool offering benchmark data, provided in a platform, which would allow gauging and improving the effectiveness of security defence algorithms. To this end in this paper we introduce a platform that allows one to generate large parametrized datasets of simulated Internet traffic consisting of the combination of attack-free and malicious network traffic patterns. For the simulations we use the ns3 discrete-event network simulator. To make the resulting dataset appropriate for intrusion detection system benchmarking purposes we investigate the statistical characteristics of normal and intrusive traffic patterns. Finally we present a use case in which we validate our results.

Keywords: Network Traffic Simulation; Intrusion Detection; DDOS; NS-3

1 Introduction and Background

Internet traffic simulation has long been important for network intrusion detection experiments. For testing the efficiency of a newly developed algorithm, researchers are in need of an environment in which tests of an intrusion detection system can be performed. The produced dataset should contain attack-free background traffic as well as intentionally inserted malicious traffic. Many of the intrusion detection evaluation experiments have been conducted on proprietary datasets that are hard to access (due to privacy concerns) and hinder reproducible research. A great effort has been made to reduce this problem in 1998 and 1999 by MIT Lincoln Laboratory, under Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL/SNHS) sponsorship, when they created the IDEVAL benchmark datasets [2-3]. To generate a corpus they

followed the approach of recreating normal and attack patterns on a private network using real hosts, live attacks, and live background traffic. The generated data flow is similar to what can be seen between a small Air Force base and the Internet. IDEVAL has been used extensively for many years as the largest publicly available benchmark for intrusion detection system (IDS) performance evaluation, although in 2000 McHugh [4] has reported some issues about it, such as the lack of comparison of the benchmark data and real data. His suspicion that the dataset's statistical characteristics differ from live network traffic was later confirmed by Mahoney et al. [5] in 2003.

Typically there are four approaches to use background traffic in IDS testing: using no background traffic, using real traffic, using sanitized traffic and using simulated traffic [6]. When the tests are conducted without using background traffic as a reference condition, the IDS's hit rate can be determined, but nothing can be said about the false positive rate. Another drawback of this scheme is the assumption that the presence or the absence of background traffic does not change the performance of the system being analysed. Injecting attacks in real background traffic can overcome this difficulty, although, usually these experiments use a small set of victim machines, the data may contain malicious traffic or anomalies specific to the network, and even privacy concerns may arise. Sanitizing the real traffic by removing any sensitive data (for example using only the TCP headers) can reduce privacy problems, but it also can lead to unrealistic scenarios if too much data is removed, or it can unintentionally cause privacy risk if sanitization fails.

Using simulated traffic can overcome many of the above mentioned problems. It can be freely distributed without privacy concerns and it surely does not contain any unexplored attack. Another advantage is that the generated traffic can be later replayed to repeat the experiment. However, providing a testbed environment that is able to preserve all the important characteristics of real life Internet traffic is a great challenge. In [7] Floyd and Paxon present a detailed description of these simulation difficulties that are mainly present due to the heterogeneity and the rapid change of the Internet.

In the literature we can find basically two ways of network traffic generation. One of them is trace-based generation, where the generated traffic is the replication of some previously recorded real traffic traces. Generators like this for example are TCPReplay [8] and TCPivo [9]. The other solution is the analytical model-based approach. It this scheme, the generation is based on statistical models. Some widely used solutions like this are:

Traffic Generator (TG) that is capable to generate constant, uniform, exponential on/off UDP or TCP traffic [10].

MGEN is both a command line and GUI traffic generator. It provides programs for sourcing/sinking real-time multicast/unicast UDP/IP traffic flows [11].

RUDE/CRUDE: RUDE stands for Real-time UDP Data Emitter and CRUDE for Collector for RUDE. RUDE is a small and flexible program that generates traffic to the network, which can be received and logged on the other side of the network with the CRUDE. Currently these programs can generate and measure only UDP traffic [12].

Distributed Internet Traffic Generator (D-ITG) is a platform capable to produce traffic that accurately adheres to patterns defined by the inter departure time between packets and the packet size stochastic processes [13].

Internet Traffic Generator (ITG) allows the reproduction of TCP and UDP traffic and to accurately replicate appropriate stochastic processes for both Inter Departure Time and Packet Size random processes. ITG achieves performance comparable to that of RUDE/CRUDE, but additionally it makes available a greater number of traffic source types [14].

2 Our Network Traffic Generator

Our goal was to build a parametrizable tool that is able to generate realistic attack free HTTP traffic combined with DDOS attack traffic. To this end, we used the widely known NS3 event based network traffic generation tool as the basic environment. Since there is no generally accepted definition about how HTTP traffic has to be like, different servers have different user habits (e.g. Facebook is checked several times a day, while news are usually read in work after lunch) we tried to make our solution as configurable as possible.

Our background traffic generation model relies on the work of Choi and Limb [15]. This is one of the most wildly used traffic generation models. According to their study, a web browsing user can be described by an on-off process. The “on” stage starts as soon as the user requests a web page. Upon a request the main object is downloaded that contains the basic structure of the web page and the links to inline objects. The main object is followed by the inline objects that actualize the embedded content of the page, these can be scripts, images, etc. The “on” stage ends when all the elements of the requested page are downloaded. After that, a silent “off” stage takes place while the user is reading the retrieved content. Although the basic concept is still viable, the exact measurements of HTTP traffic made by Choi and Limb are considered obsolete because since the time of their research the nature of web traffic went through a significant change. The appearance of social networks and multimedia streaming, the more complex structure and the new services of web pages required new measurements to better describe web browsing behaviour. The actual measurements we relied on for DDOS detection system evaluation in Section 3 were conducted by Pries et al. in [16]. The presented results are based on the top one million visited web pages. The settings that we used are presented in Table 1.

Table 1
HTTP model parameters

Parameter	Best fit
Main object size	Weibull (28242.8,0.814944)
Number of main objects	Lognormal $\mu = 0.473844$; $\sigma = 0.688471$
Inline object size	Lognormal $\mu = 9.17979$; $\sigma = 1.24646$
Number of inline objects	Exponential $\mu = 31.9291$
Reading time	Lognormal $\mu = 0.495204$; $\sigma = 2.7731$

Several calculations have been made about the content of the configuration, as we wanted to keep it simple, yet customisable enough to fulfil any need of HTTP traffic. The resulted bunch of options is detailed in Table 2.

Table 2
Simulation parameters

numOfNodes	The number of nodes in the simulation, this includes the server and the dos clients as well
numOfDosClients	The number of nodes that will generate DoS traffic instead of general client behaviour
startTime	This parameter shows in which simulated second the clients start their work (does not affect DoS clients)
endTime	Which simulated second the clients stop their work
dosStartTime	Which simulated second the DoS clients start their work
dosStopTime	Which simulated second the DoS clients stop their work
serverPort	Number of port which the server listens on, and clients connect to
dataRate	Global link speed
delay	Number of seconds required for the first bit of a packet to arrive at the destination host
packetLoss	The probability that a packet gets lost while transferring
inlineObjectSizeLogNormalVariable	Due to studies [16] the number of inline objects in a mainline object is a Lognormal random variable. This variable's two parameter can be set in.

npageRequestsLogNormalVariable	Due to studies [16] the time between two mainline object requests is a Lognormal random variable. This variable's two parameter can be set in.
dosClient/ pageRequestsLogNormalVariable	The same as above, but it refers to DoS clients
avgClicks sleepTimeBetweenClickings	These parameters work together. Legal clients after the set number of average clicks will not request any other object for an average of SleepTimeBetweenClickings seconds.

A key in our data generator is to make it easy to use. We integrated our solution into a minimalistic web service, which results in a user friendly interface where without installing anything, a click is enough to start the simulation, and be able to download your results. This solution also, disencumbers our developer computers and puts the load to dedicated servers.

Since ns3 compiles for quite a long time, we wanted to give a solution which does not require recompilation too often but also stays parametrizable. Not just to fasten the simulations but to provide an easy summation of possible parameters, we implemented a configuration parser, which works with an XML file, and therefore does not require recompilation after a parameter changes.

2.1 Case Study

To show that our solution is easily configurable to generate any type of network traffic we simulated our local web server, and generated simulated data with the attributions of the original. In this section we are going to discuss the details of this experiment.

To gather a fair amount of real network data we monitored all the packets received by our local web server, for about a month-long period. Figure 1 shows the frequency of page requests that arrived at our server during the monitored time interval.

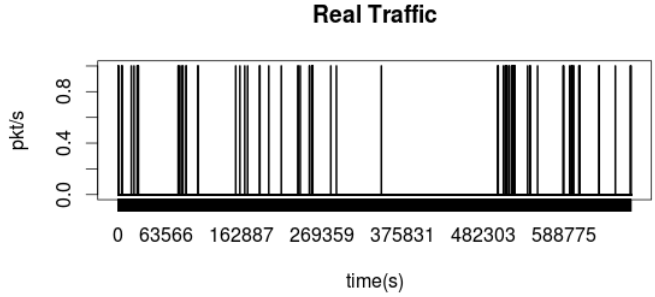

Figure 1
Main object requests per second in the real data

To estimate the necessary parameters for the simulation we filtered the data for the http requests. We used R's "fitdstrplus" library to give the best estimation for the distribution of the time between two main object requests, the average number of consecutive main object requests and for the "sleepTimeBetweenClickings" that corresponds to the time between two visits of the same client to the web page. The results are presented in Table 3.

Table 3
Parameter estimates

Parameter	Best fit
MainpageRequests	Exponential $\mu = 857.5256$
avgClick	Pareto mean = 5; shape = 1.1; bound = 20
sleepTimeBetweenClickings	Exponential $\mu = 20696.83$

Figure 2 shows the frequency of the page requests in the simulated traffic with the above configuration.

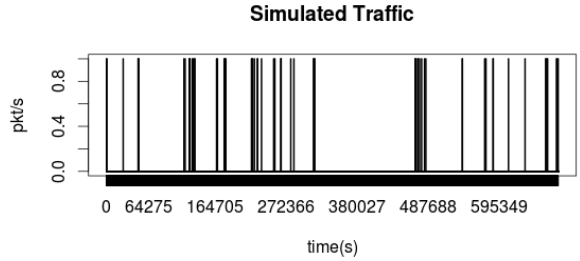

Figure 2
Main object requests per second in the simulated data

2.1.1 Self-Similarity and Long-Range Dependence

The distribution of traffic on the Internet commonly exhibits self-similarity. The first observation of the phenomena was made by Leland and Wilson [17]. They commented in detail on the presence of "burstiness" across an extremely large range of time scales in the data collected at Bellcore Morristown Research and Engineering Center on several Ethernet LANs. The first statistically rigorous analysis was made in 1994 by Leland et al. [18] on the same dataset. Their research pointed out that the Ethernet LAN traffic is self-similar, irrespective of where and when the data were collected in the network. They showed that the Hurst parameter better describes the fractal-like nature of the traffic and able to capture its "burstiness", when other methods (the index of dispersion, the peak-to-mean-ratio or the coefficient of variation for inter arrival times) fail to do so. Their observations have been supported by later research and their findings have led the research community to make significant efforts towards developing appropriate mathematical and statistical techniques that provide a network-related

understanding of the observed self-similar scaling behaviour [19-26]. A mathematical description of self-similarity used by Bai and Shami in 2013 [27, 28] for network traffic simulation is the following:

Let X_i (i = 1, 2, …) be an increment process and $X_j^{(m)}$ (j = 1, 2, …) be another process, which is obtained by averaging the values in non-overlapped blocks of size m in X_i, i.e.:

$$X_j^{(m)} = \frac{1}{m}\left(X_{jm-m+1} + X_{jm-m+2} + \ldots + X_{jm}\right). \quad (1)$$

The process X_i is said self-similar if $X_j^{(m)}$ is similar in distribution to $m^{H-1}X_i$, where m ($m \geq 1$) is the scale parameter and H is the Hurst exponent. In a more understandable form it implies:

$$\mathrm{Var}\left(X_j^{(m)}\right) = m^{2H-2}\,\mathrm{Var}\left(X_i\right). \quad (2)$$

Another wildly researched characteristic of network traffic is long-range dependence (LRD). Its discovery in network traffic, has fundamentally changed the conventional wisdom by stating that the correlation of packet inter arrivals decays slower than in traditional traffic (e.g., Markov) models. LRD and self-similarity are strongly related. From the definitions of [29] we use the inference that LDR characterizes a time series if it holds for the Hurst exponent H that $0.5 < H \leq 1$. As H approaches 1, the dependence becomes stronger.

During background traffic generation we found it important to test if the simulated data preserved the LRD characteristics of the real data. To test LRD of the packet arrivals we estimated the Hurst exponent using the aggregated variance method of the fArma R library. The estimation of the real and the simulated traffic's Hurst exponent satisfied our expectations since it resulted similar numbers, 0.57 and 0.52 respectively.

3 DDoS Detection

3.1 Brief introduction to DDoS Attacks

Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks are attempts to make machines, and network-related resources or services (e.g. Webserver) unavailable for its legal users.

DDoS attacks are launched by more than one hosts, but more often the attacker uses botnets (network of zombie hosts, infected by some kind malware) and thousands of hosts from all around the world. According to [30], in 2014 the frequency of recognized DDoS attacks had reached an average rate of 28 per hour.

In this paper we focus on HTTP GET flood attacks [31]. It is an application layer attack and in this case the attacker's hosts send seemingly legitimate HTTP GET requests to the webserver that they aim to attack. These attacks do not use malformed packets, or spoofing, and they require less bandwidth than other attacks to bring down the targeted servers, but they require some understanding of the targeted application. This type of attack is harder to detect than some others, mainly because it uses valid HTTP GET requests, and it usually doesn't generate significant network traffic. These types of measurements don't really affect DDoS detection, that's why we used the frequency of the page requests in our detector.

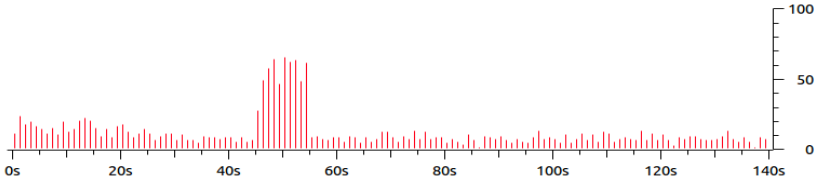

Figure 3

Packet per second rate of a generated DDoS attack

Figure 3 shows the packet per second rate of a generated DDoS attack. In this case, we used the recorded traffic (~100 MB) of the server that was attacked by some malicious clients between time 45 and 55, which is illustrated by higher lines in the plot. The traffic before, and after the DDoS is normal.

3.2 Detecting DDoS attacks with Snort

Snort [32] is widely used, free and open source network intrusion detection software. It is capable of real time traffic analysis and packet logging, and because of its huge community, it is one of the most widely developed intrusion detection systems in the world.

Snort has a set of rules defined by the user, and the network traffic is analysed against these rule-sets. After the detection special actions can take place based on what has been identified.

Snort has a built-in set of rules for DDoS detection, which could be used for validation, but these rules were very generic, so we tried another approach.

Our approach consists of two parts:

- Analysis of a training set of normal traffic data
- Use the parameters from phase 1 to detect DDoS attacks with Snort

In the 1st phase we analysed normal traffic data for parameters which will be used for detection. In this case we used only the packet/sec rate of the most active client from the most loaded moment of the server. To achieve this we used a python and the "dpkt" module (python-dpkt package on ubuntu 14.04) for "pcap" analysis.

In the 2nd phase we used the parameter from above as a base threshold for a rate limit-like approach. Furthermore we used a multiplier for the parameter, but its value was decided by empirical methods for every unique case. We generated rules for Snort, and after we started it with the given ruleset, Snort raises alerts if any of the clients reaches the packet rate.

In the case of datasets larger than many gigabytes, it is hard to check the generated traffic in the way we have done it in the case of Figure 3, so instead of wireshark, we used Snort to check for DDoS attacks. Snort was able to analyze these pcap files in reasonable time, with the method mentioned above, so we could easily validate our generator.

Conclusions

We deeply studied the different types of web traffic, with outstanding attention to the relation between HTTP and the many DoS/DDoS attacks. Simulations are much cheaper, quicker and easier to use than real systems, so we considered the need of a tool, which is able to generate valid and parametrisable HTTP traffic, with customisable DDoS attackers.

We extended the existing NS3 with our classes, and implemented our own XML configurable simulator and a webservice to make our simulator easy to use.

The simulator is perfect for generating hundreds of GB traffic. Since such volume of data cannot be verified manually, we implemented a python script that is able to generate Snort rules.

References

[1] CyberEdge: 2014 Cyberthreat Defence Report for North America & Europe, a CyberEdge report sponsored by ForeScout Technologies, Inc., 2014

[2] R. Lippmann, et al.: The 1999 DARPA Off-Line Intrusion Detection Evaluation, Computer Networks 34(4) 579-595, 2000. Data is available at http://www.ll.mit.edu/IST/ideval/

[3] Lippmann, Richard P., et al.: Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation, DARPA Information Survivability Conference and Exposition, 2000, DISCEX'00, Proceedings. Vol. 2, IEEE, 2000

[4] McHugh, John: Testing Intrusion Detection Systems: a Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. ACM Transactions on Information and System Security 3.4 (2000): 262-294

[5] Mahoney, Matthew V., and Philip K. Chan: An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2003

[6] Hu, Vincent, et al.: An Overview of Issues in Testing Intrusion Detection Systems (2003)

[7] Floyd, Sally, and Vern Paxson: Difficulties in Simulating the Internet. IEEE/ACM Transactions on Networking (TON) 9.4 (2001): 392-403

[8] URL http://tcpreplay.synfin.net/

[9] URL http://www.thefengs.com/wuchang/work/tcpivo/

[10] URL http://www.postel.org/tg/

[11] URL http://www.nrl.navy.mil/itd/ncs/products/mgen

[12] URL http://rude.sourceforge.net/

[13] Avallone, Stefano, et al.: D-ITG Distributed Internet Traffic Generator. Quantitative Evaluation of Systems, 2004, QEST 2004, Proceedings, First International Conference on the. IEEE, 2004

[14] Avallone, Stefano, Antonio Pescape, and Giorgio Ventre: Analysis and Experimentation of Internet Traffic Generator. Proc. of New2an (2004): 70-75

[15] Choi, Hyoung-Kee, and John O. Limb "A Behavioral Model of Web Traffic."Network Protocols, 1999 (ICNP'99) Proceedings Seventh International Conference on. IEEE, 1999

[16] Pries, Rastin, Zsolt Magyari, and Phuoc Tran-Gia "An HTTP Web Traffic Model based on the Top One Million Visited Web Pages." Next Generation Internet (NGI) 2012 8th EURO-NGI Conference on. IEEE, 2012

[17] Leland, Will E., and Daniel V. Wilson: High Time-Resolution Measurement and Analysis of LAN Traffic: Implications for LAN Interconnection, INFOCOM'91, Proceedings, Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s, IEEE, 1991

[18] Leland, Will E., et al.: On the Self-Similar Nature of Ethernet Traffic (extended version) Networking, IEEE/ACM Transactions on 2.1 (1994): 1-15

[19] M. E. Crovella and A. Bestavros: Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, ACM SIGMETRICS, Vol. 24, No. 1, pp. 160-169, 1996

[20] P. Barford and M. Crovella: Generating Representative Web Workloads for Network and Server Performance Evaluation, ACM SIGMETRICS, Vol. 26, pp. 151-160, 1998

[21] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella: Changes in Web Client Access Patterns: Characteristics and Caching Implications, World Wide Web, Special Issue on Characterization and Performance Evaluation, Vol. 2, pp. 15-28, 1999

[22] C. R. Cunha, A. Bestavros, and M. E. Crovella: Characteristics of WWW Client-based Traces, 1995, technical Report TR-95-010, Boston University Computer Science Department

[23] T. Ott, T. Lakshman, and L. Wong: SRED: Stabilized RED, in IEEE INFOCOM, 1999, pp. 1346-1355

[24] P. Barford and M. E. Crovella: A Performance Evaluation of HyperText Transfer Protocols, in ACM SIGMETRICS, 1999, pp. 188-197

[25] Paxson, Vern, and Sally Floyd. "Wide Area Traffic: the Failure of Poisson Modeling." IEEE/ACM Transactions on Networking (ToN) 3.3 (1995): 226-244

[26] Riedi, Rudolf H., and Walter Willinger.: Toward an Improved Understanding of Network Traffic Dynamics. Self-Similar Network Traffic and Performance Evaluation (2000): 507-530

[27] Bai, Xiaofeng, and Abdallah Shami.: Modeling Self-Similar Traffic for Network Simulation. arXiv Preprint arXiv:1308.3842 (2013)

[28] P. Orenstein, H. Kim and C. L. Lau: Bandwidth Allocation for Self-Similar Traffic Consisting of Multiple Traffic Classes with Distinct Characteristics, IEEE GLOBECOM 2001, Vol. 4, pp. 2576-2580, December 2001

[29] Karagiannis, Thomas, Mart Molle, and Michalis Faloutsos.: Long-Range Dependence Ten Years of Internet Traffic Modeling. Internet Computing, IEEE 8.5 (2004): 57-64

[30] Preimesberger, Chris (May 28, 2014) "DDoS Attack Volume Escalates as New Methods Emerge". *Eweek*

[31] Incapsula - DDoS Attack Glossary – HTTP Flood http://www.incapsula.com/ddos/attack-glossary/http-flood.html

[32] Snort: - URL: https://www.snort.org