DDoS Network Flow Forensics Analyser

Cristian Turetta, and Andrea Perazzoli

Abstract—Distributed denial-of-service (DDoS) is a rapidly growing problem. The multitude and variety of both the attacks and the defence approaches is overwhelming. In this report we propose a tool which analyse, using big data framework, recorded traffic and by using statistical tools like standard deviation and difference from mean, it tries to detect the set or at least a subset of attackers.

1 Introduction

Denial of Service (DoS) attack is launched to make an internet resource unavailable often by overwhelming the victim with a large number of requests. DoS attacks can be categorized on the basis of single source and multi source. Multi source attacks are called distributed Dos or DDoS attacks [1].

There are various type of tool in order to detect and deal with DDoS attacks, these tools can be applied in real time, such as intrusion detection systems (IDS) or by analysing network flow records offline doing a forensics analysis, which is our focus. Computing forensics analysis over a recorded network flow can be useful, we may understand if someone is trying to flood a network to get a denial of service and eventually recognise it. Evidence of such intrusions is required in case the affected wants to pursue the court and legal action is to be taken against the adversary.

Forensics investigations are not trivial to accomplish and often done manually, this because the attacker can mask its attempts by mixing legitimate requests with malicious ones.

DDoS attacks aim to compromise the availability of a system or a network, the attack is launched by the adversary which has take control over bots, compromised machines connected to internet, that sends several requests to the victim and overwhelm it with large amount of traffic. This creates a bottleneck and the victim can no further deal with this traffic denying service to them.

During DDoS attacks, the log files swell up to huge sizes, these log files if analysed properly and effectively can help detect and recover from a DDoS attack [1]. Log files can take a long time if processed through conventional means thus we decide to use big data's tool and framework in order to get a faster processing and investigation.

In this report we present our tool which uses *Pig-latin* script embedded into a *Python* program that can be used to analyse network log file, `pcap` format [2], and returns statistical information about the recorded traffic, since there are a lot of attack variety, we focus on one of the principals which is UDP flood. It can be initiated by sending a large number of UDP packets to random ports on a remote host.

This report is structured as follows, in Section 2 we present the statistical tool used in our analysis. In Section 3 we present the project structure and implementation. In Section 4 we focus on analysis results. In Section 5 we discuss the performance of our tool in terms of computational time and resources. Section 6 concludes the report with our considerations.

2 Mathematical Model

Based on data volume rates, we do need mathematical models to identify which are the source addresses that may be proponents of an attack. Since we categorise data, using big data framework in order to analyse log files and generate, for simplicity we explain each field by referring to a single entry which represents an exchange between a source ip and server.

- *n_packets*
  Represents the amount of packets.
- *total_volume*
  It is the sum of all packets length.
- *time_difference*
  It is the time difference between the first communication and the last one, we consider it as a time window.
- *ratio_vol_td*
  Represents the volume over seconds exchanged during the time window.

The mathematical model must be able judge the data [3] by using a threshold, in our case the maximal volume of traffic per second supported by the server under analysis represents this threshold. This threshold can be estimated by adding an α factor to the mean of volume exchanged during the time of records.

Using *python* in order to manipulate the results returned by *Pig-latin* we decide to use **standard deviation** as mathematical model to identify suspicious source address. The formula for the sample standard deviation is

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{1}$$

where $x_1, x_2 \ldots x_n$ are the *ratio_vol_td* values grouped by source and destination ip, $\bar{x}$ is the mean volume exchanged and N is the number of sample in the dataset generated by *Pig-latin* script. We decide to output three plots, *data analysis* which figure out the amount of data, in terms of megabyte, exchanged between a source and the server. *Volume analysis* is like the previous one but represents the volumes per second exchanged between a source and the server, we use these two plots together to identify the data flow and thanks to these attackers may be detected more easily. Then we plot also the standard deviation.

3 Project structure and implementation

Our project and its source code is freely downloadable on Github. Here, we focused on the UDP flood D(D)oS analysis of pcap records: the goal is to point out good and evil users given a pcap network sniff file converted to csv format. The project is a multi-layered tool which primarly consists in two executable Python 3 scripts:

3.1 DDoSAnalysis

DDoSAnalysis.py can be used in two different modes, depending on the line parameters used:

- *-g dataset_name n_members n_lines n_attackers atk_volume atk_duration*
  Generates a random, bogus dataset in the current working directory with the name specified in the second argument, alongside with the number of normal network users specified in *n_members* argument, the dimension of the dataset (in lines), the number of infected machines, the attack volume (per packet) and its duration. At the end of the generation process, it copies the dataset into the Hadoop File System. We assumed that Hadoop is installed and a folder tree under `hdfs://user/your_user/project/input` exists.
- *-a dataset_name*
  Begin the analysis of the dataset *dataset_name* using a Pig script. In order to work, the dataset must have been previously copied into the Hadoop input folder, which automatically happens if the dataset is generated using *-g* option. It saves the elaborated dataset under `outputs/dataset_name` with an image consisting of a plot of every agent average velocity (Mbps), an attack volume graph and a statistical image which shows the squared margin to mean velocity.
- *-anp dataset_name*
  It's like the *-a*argument, but it doesn't start Pig analysis. It's used when we have already completed a Pig analysis, and we only want to aggregate data and obtain the plots.
- *-ga dataset_name n_members n_lines n_attackers atk_volume atk_duration*
  Launches both the generation and the analysis
- *-sga dataset_name n_members n_lines n_attackers atk_volume atk_duration*
  Launches both the generation and the analysis. After generation but before the Pig analysis, it prints an estimation of the dataset dimension, letting the user choose whether to proceed or not.

The script also automatically records infos about performance timing under `PerformanceHistory.csv`.

3.2 PerformanceAnalyser

PerformanceAnalyser.py is used to automatically plot all the infos stored under `PerformanceHistory.csv`. It supports two modes:

- *-a img_name*
  Stores a plot under the current working directory named *img_name* of analysis statistics (history of dataset analyzed and time elapsed)
- *-g img_name*
  Stores a plot under the current working directory named *img_name* of dataset generation statistics (history of dataset generated and time elapsed)

PerformanceAnalyser.py also exposes a method used as a wrapper to call the generation and analysis routines, using a **CProfile** python module to gain time statistics.

The first two mentioned scripts are the user interface of our tool. However, we have other core scripts which make the generation/analysis possible:

3.3 DatasetGenerator

DatasetGenerator.py contains the core generation routine of datasets. It generates a pool of innocent IPs and an attackers' one, then it fills line-by-line the dataset with random and bogus informations, extracting random users and attackers. It saves a csv file in the format: *id, time, source_ip, dest_ip, protocol, packet_size, payload*

3.4 Evaluator

Evaluator.py exposes the main routine which processes the Pig script output. It computes the mean velocity of all users, and then produces a plot consisting of the velocity of every single user, represented with a blue line, previously calculated by the Pig script (*udpfloodpcap.pig*) and the mean velocity of all users, represented with a red line. The data scientist could distinguish between evil and good users just looking at the deviation from the average. It also plots a volume comparison graph between users and the squared margin to mean velocity.

3.5 Pig script

Udpfloodpcap.pig calculates the mean velocity of every machine given a dataset in input. This is the core of the analysis tool: it uses Pig Latin mapreduce paradigm. The script loads the dataset, filter the records having UDP protocol, and group them by *(source_ip, destination_ip)* tuple. At this point we have to handle a list of *map(tuple, bag)* containing all the corresponding packets sent by a machine. We can then calculate the number of packets (counting the elements in the bag), the total volume exchanged by a particular machine (summing up the corresponding data for each element of the bag) and the mean velocity in bps (using the total volume divided by the max time minus the min time). This is a crucial factor which we use to discriminate good and evil users.

REFERENCES

[1] Rana Khattak, Shehar Bano, Shujaat Hussain, Zahid Anwar. *DOFUR: DDoS Forensics Using mapReduce*. Frontiers of Information Technology, 2011.
[2] Wireshark Wiki, Development. Last access *May 15 2019*. https://wiki.wireshark.org/Development/LibpcapFileFormat
[3] Theerasak Thapngam, Shui Yu, Wanlei Zhou, S. Kami Makki. *Distributed Denial of Service (DDoS) detection by traffic pattern analysis*. Springer Science, 2012.