Botnet IDS Using Clustering Analysis of Network Traffic

An essay submitted in partial fulfillment of

the requirements for graduation from the

Honors College at the College of Charleston

with a Bachelor of Science in

Mathematics, Data Science

KAYA TOLAS

MAY 2017

Advisor: Paul Anderson

Botnet IDS Using Clustering Analysis of Network Traffic

Kaya Tollas
College of Charleston, Department of Computer Science
tollask@g.cofc.edu

1. Introduction

IoT, or Internet of Things, is expanding at an increasing rate and setting us up to be surrounded by smart devices. However, along with its enormous potential for positive change, IoT also increases threats to our privacy and security. Security is not usually a design feature of smart devices, many of which are designed by manufacturers with no domain knowledge in cybersecurity. As of now, most devices that connect to the internet can be hacked. This lack of IoT "public health" is the manufacturing sector is compounded on the user side, as users rarely change default passwords or reboot their devices.

Bots, or remotely-controlled devices connected to the internet, are increasingly infiltrating home networks as smart devices (e.g. light bulbs, refrigerators, thermostats) become more prevalent in homes. Bots, or "zombie devices", communicate with their remote hosts and receive commands to initiate activities such as participating in DDoS (Distributed Denial of Service) attacks, sending spam, or mining bitcoins. While botnet structures, activities, and network protocols can be very diverse and sophisticated, all botnets are distinguished by their use of command and control (C&C), the method used to coordinate bot activities and maintain communications with the remote botmaster. It is relatively simple to hack IoT devices, many of whose default security settings can be found online, and turn them into bots. Now that home networks are hosting increasing numbers of devices, they can serve as breeding grounds for sub-botnets.

In order to improve the poor state of IoT public health and practices today, customers can implement their own solutions until security and privacy are the standard among manufacturers. As the potential for intruders to penetrate more personal networks increases, it is important that users have an idea of what is happening on their network.

My contribution to this effort is a clustering algorithm for network flows that can operate alongside any home network. The goal of this algorithm is to cluster network flows into suspected sub-botnets and normal flows. This algorithm is inspired by the BotMiner framework by Gu et al [1], which was designed to cluster botnets with no a priori knowledge of bot behavior and botnet structure and protocol. Since botnets are evolving quickly and can already be quite sophisticated, my algorithm maintains the level of generality of the BotMiner framework.

The main goal of this research was to produce a simple, open-source implementation of a network flow clustering algorithm that can be extended and used for further research in

intrusion detection systems for botnets. The code for this project is on Github [2].

2. Clustering Algorithm Framework and Implementation

2.1 Program requirements

In order to run as intended, this algorithm requires that the user install a packet analyzer such as tshark, Wireshark's command line tool [3], to collect and log their network flows in the form of .pcap files. Using tshark, users can efficiently filter and log only relevant fields for the algorithm, namely, packet length in bytes, relative of capture, source and destination IP addresses, and source and destination ports (TCP/UDP)**.** Users can also test the algorithm using pcap files available online, such as Wireshark's library of sample captures [4] and any available captured botnet traces. I obtained the botnet traces for this algorithm from the University of New Brunswick, whom I contacted for access to their botnet dataset [5]. The code is in Python 3, and makes use of Python's pandas, sklearn, numpy, datetime, csv, math, and matplotlib's pyplot packages.

2.2 Flow collection and filtering

The initial stage of the algorithm is the collection and filtering of network communication flows. As mentioned earlier, the fields needed by the algorithm are the source IP and port, destination IP and port, relative times in the capture, and number of bytes per packet. Using tshark, users can set these filters using the command

```
$ tshark -nr your-capture.pcap -T
fields -E header=y -e
frame.time_relative -e frame.len -e
ip.src -e ip.dst -e tcp.srcport -e
tcp.dstport -e udp.srcport -e
udp.dstport  > yourPCAP.txt
```

This creates a tab delimited file from your .pcap file, which is then ready for analysis in Python. It is worth noting that although Wireshark allows for packet payload inspection, we do not consider this data in the algorithm as it is useless when payloads are encrypted and is extremely memory-intensive to store and use.

2.3 Flow aggregation into C-flows

This concept is borrowed directly from Gu et al in their Botminer paper [1]. The algorithm filters and aggregates network flows into "C-flows", which is a data structure of records from the same source IP and port to the same destination IP and port. The grouped C-flows then tell us "who is talking to whom" in a given capture. From this data structure we can calculate statistical representations of each C-flow so that we can subsequently perform mathematical clustering analysis.

2.4 C-flow vectorization and smoothing

Each C-flow is then represented by a vector of four flow statistics, as seen in Figure 1.

1. *bytes per packet (bpp)*: total sum of bytes divided by total number of packets sent in flow
2. *bytes per second (bps)*: total sum of bytes divided by duration of the flow
3. *packets per second (pps)*: total packets in flow divided by duration of flow
4. *packets per flow (ppf)*: total packets in flow

Since these variables can range widely, each C-flow is then standardized to further prepare it for clustering analysis. This also borrows from Gu et al.'s binning technique [1], which generates a k-bin for each feature (bpp, bps, pps, ppf), with k = 13 quantile values ($q_{5\%}$, $q_{10\%}$, $q_{15\%}$, $q_{20\%}$, $q_{25\%}$, $q_{30\%}$, $q_{40\%}$, $q_{50\%}$, $q_{60\%}$, $q_{70\%}$, $q_{80\%}$, $q_{90\%}$, q_{max}) where $k_1 = q_{5\%}$, $k_2 = q_{10\%}$, and so on.

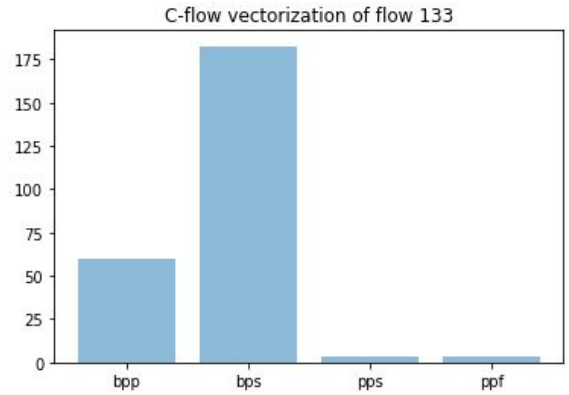

Figure 1. Original vectorization of C-flow

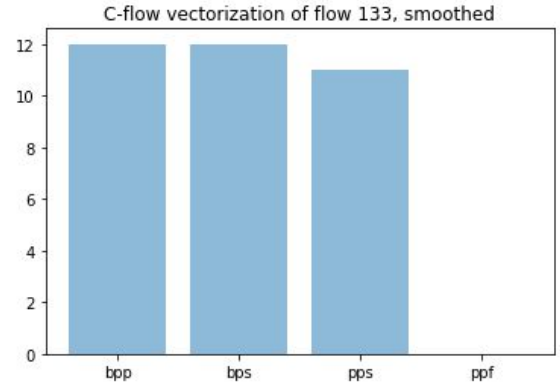

Figure 2. Smoothed vectorization of C-flow

This is calculated on vectors of feature values (i.e. bpp, bps, fps, ppf) from all C-flows, so that for each feature, each C-flow now has a value of range 0-12 corresponding to its quantile as compared to the other C-flows, as seen in Figure 2.

2.5 Clustering

The standardized C-flow vectors are then clustered using Python's scikit-learn (sklearn) K-means clustering algorithm. This clusters K similar groups based on a specified K number of means. The goal is that C-flows containing bots will be clustered together into sub-botnet clusters, and regular C-flows will be clustered separately from bot clusters. Since I trained my algorithm using labeled botnet capture data from New Brunswick University, I was able to test the effectiveness of my clustering algorithm using an entropy criterion. Upon running the algorithm with a range of different means, I found my clustering accuracy was optimized at 9 clusters. I measured clustering accuracy by calculating Shannon entropy, an information criterion based here on the proportion of bot IPs and non-bot IPs in each cluster.

2.6 Cluster reports

The output of the algorithm is a data frame of C-flows and each cluster they belong to. Using this, users can view which flows are behaving similarly and anomalously, and can reboot or change the passwords on devices with IP addresses flagged by the algorithm.

References

[1] G. Gu, R. Perdisci, J. Zhang, W. Lee. "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection". College of Computing, Georgia Institute of Technology. 2008. https://www.usenix.org/legacy/event/sec08/tech/full_papers/gu/gu_html/

[2] K. Tollas. C-flowClustering (2017) https://github.com/kayat95/C-flowClustering

[3] tshark: Terminal-based Wireshark https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html

[4] Wireshark: SampleCaptures https://wiki.wireshark.org/SampleCaptures

[5] ISCX Botnet dataset. University of New Brunswick http://www.unb.ca/cic/research/datasets/botnet.html