Feature Extraction and Anomaly Detection in Network Traffic (PCAPs)

Background

My study is based on a 2019 study by Yang et al. entitled *Feature Extraction for Novelty Detection in Network Traffic*. The main idea from that study is PCAP files were partitioned into “forward flows” using a five-tuple identifier (IP Src, Src Port, IP Dst, Dst Port, IP Proto), and then further partitioned into “subflows” based either on fixed time intervals or a timeout period (time between packets, or inter-arrival time). Features were extracted from the subflows and used to train and test several machine-learning methods, including an autoencoder, for anomaly detection.

Setup

Yang et al. used a tool called netml to partition the flows. However, I had difficulty getting the tool to work as indicated on my 5G PCAPs. I was forced to replicate the functionality. While netml relies on Scapy to handle PCAP files, I wrote a shell script that extracts fields and saves them to a CSV using tshark. To replicate the study, the following raw fields are extracted from each packet:

- Frame: time_epoch
- IP: len, proto, src, dst, ttl
- TCP: srcport, dstport, flags
- UDP: srcport, dstport

The CSV file is loaded into a Pandas data frame where partitioning and feature extraction occur. It should be noted that a PyShark alternative to the tshark script was explored, but it was much slower.

See Methodology.pdf for more information on the extraction process.

Data Preprocessing (CSV)

Since I only wanted IP packets, I dropped all rows with NAN values in the ip.proto column. I converted the remaining NAN values to 0 to account for ICMP packets having no TCP ports. This also sets TCP flags to 0 for packets using UDP. The srcport and dstport columns are set to the max of their TCP and UDP counterparts, and those columns are dropped. For CSV files created through concatenation of other CSV files, duplicate field header rows are removed. Columns are then converted from objects to an appropriate data type: strings for ID columns, datetime for time epochs, and integers for the rest. Note that base 16 was specified when converting TCP flags.

Flow Partitioning

The main idea is to group packets together based on some chosen identifiers. The study used a 5-tuple to define *forward flows* (ip.src, srcport, ip.dst, dstport, ip.proto). However, we can partition the flows any way we like. For example, I have been experimenting with partitioning flows by (ip.dst, dstport, ip.proto) because I believe it will improve DDoS detection. I define these as *inward flows*.

To partition the flows, a data frame is created using only the identifier columns. The drop_duplicates() function is called to obtain unique identifiers. The identifiers are iterated and, using multiple column conditions, all matching rows (packets) are selected from the original dataframe. The raw feature columns from these rows (frame.time_epoch, ip.len, ip.ttl, tcp.flags) are selected and the collection is saved as a partition.

A dictionary links identifiers to partitions (flows). The identifiers are converted to strings for use as keys in the dictionary. Partitions with less than 2 packets are dropped as this is the minimum for flow analysis. This is also true for subflow partitioning.

Subflows

To obtain more samples and to better capture the general behavior of our network traffic, flows must be sub-partitioned into "subflows." Two methods are available: timeout period and fixed time interval. I found that neither method results in significantly more samples for my 5G data. Careful selection should be made, however, as some network traffic does not have sufficient breaks for the timeout method to work.

Timeout Period

To split flows by timeout, I used Pandas' diff() function on the frame.time_epoch column of each flow. This function returns a series containing the difference between each row and its preceding row, including datetime columns. Indices are saved for rows with a (inter-arrival time) difference greater than the timeout interval.

The saved indices are iterated and, using Pandas' loc() function, subflows are selected from the flow and saved. If there are no sub-partitions, the entire flow is saved as a subflow.

Fixed Time Interval

To split flows by fixed time interval, the index and value of the first frame.time_epoch in each flow are selected as starting points. A subframe is created containing all time epochs from the starting index to the end of the dataframe, and the start time is subtracted from it. The last index of the subframe <= our fixed time interval is recorded as the end of the subflow. The first index of the subframe > our fixed time interval is recorded as the start of the next subflow, and the process repeats.

The recorded indices are processed in a similar way as the timeout interval indices, with a few minor differences that can be observed in the code. If a flow has no sub-partitions, the entire flow is saved as a subflow.

Feature Selection (Subflows)

Yang et al.'s study was comprehensive, trying several approaches to feature selection. I felt most comfortable with the statistical representation of subflows. Ten statistical features were originally selected: subflow duration, packets/sec, bytes/sec, and packet size information (mean, std, q1, q2, q3, min, max). I dropped subflow duration because I believe if the autoencoder is trained to associate statistics with specific durations, more false positives will occur. A real-time system should verify that current network statistics are within nominal range, independent of analysis duration. I also converted bytes/sec to bits/sec because this is a more standard representation of mobile network traffic rates.

I added some TCP stats (SYN/sec and average TTL) because Yang et al. noted that adding header information may help with novelty detection in some cases (e.g., a SYN flood or change in network topology). However, I don’t like how they represented that information in the feature vector and am hoping my “flags/sec” and average TTL ideas are sufficient. If not, the data representation can be modified according to their study (with some effort). Yang notes that TTL may not be a useful feature if it is the same for all devices on the network.

Autoencoder

An autoencoder is chosen for its ability to be trained without labeled data. The autoencoder is similar to the one used by Yang et al: input layer of size *d*, hidden layer of size *d-1*, latent layer of size $\lceil d/2 \rceil$, hidden layer *d-1*, and output layer *d*. These are the default dimensions and can be optimized according to the study. Yang et al. use the LeakyReLU activation function and Adam optimizer. Both of these options are available for experimentation. I also include options for ReLU + Linear activations and Adam enhanced with Lookahead.

Well-Posed Autoencoder

Autoencoders extend Principal Component Analysis (PCA) principles into a nonlinear space. By enforcing some of these principles, the autoencoder should become easier to tune and optimize. To test this, I include options for tied weights and unit norm constraints. Near-orthogonality also appears to be achieved with the unit norm constraint.

See PCA Principles.pdf for more information.

Training and Loading

To ensure consistency, no shuffling occurs during training (the data is already shuffled by the extractor). The best model for a configuration is saved and can be restored with the loader program. The detection threshold is set at 3 standard deviations, which includes approximately 99.7% of nominal data points.

Attack Simulation (Synthetic Flows)

The loader program allows for the generation of synthetic flows, which is useful for simulating DDoS attacks against monitored devices. To simulate an anomalous traffic flow, network statistics outside the nominal range must be generated. To accomplish this, a function was created which takes as input a packet size and number of packets per second and returns a feature vector of network statistics. These two parameters can be adjusted to control the data rate, and a data frame with any number of these attacks can be saved for testing purposes.

Each anomalous flow has equal-sized packets which makes statistics easy to calculate. The function can generate several flows at once, each with different data rates if desired. This allows us to visualize a gradient of synthetic flows with increasing data rates. Currently the gradient feature works by fixing the packets per second and changing the packet sizes. It can be modified to work the opposite way, but the idea would remain the same.

Links

Feature Extraction Study (Yang et al.): https://arxiv.org/pdf/2006.16993

Netml tool: https://github.com/chicago-cdac/netml

Autoencoder tuning (PCA): Build the right Autoencoder — Tune and Optimize using PCA principles. Part I | by Chitta Ranjan | Towards Data Science