Methodical Process

Dissection of PCAPS

- The first thing I wanted to do was view the packet fields
- I used tshark to view one of Ashley's PCAPS in XML format
    - tshark -r pw1 -T pdml > pw1.xml
    - PDML: Packet Description Markup Language
- I familiarized myself with the fields and tried to think of which would be useful
- Instead of picking random fields, I decided to research how to extract features from PCAP files
    - This is how I found Yang's PCAP study

Attempt at using netml tool

- Yang's study used a tool called netml to parse PCAP files and extract features
    - However, I was not able to get the netml tool to properly parse Ashley's PCAPS for an unknown reason
    - My only way forward was to replicate netml's functionality
- The netml tool has two parts, a PCAP parser and a machine learning tool
    - The machine learning functionality was not useful to me because I already had an Autoencoder to play with
    - I downloaded netml's parser code and began studying it

General concepts learned from netml parser code and Yang's study

- Group all packets together that are going from one machine to another
    - These are "flows" from source to destination
- Split these flows up into "subflows"
    - Fixed time intervals: Subflows are <= N seconds
    - Timeout period: Subflows are split using inter-arrival time > N seconds
- Collect information about the packets in each subflow
    - This information becomes our feature vector
    - The study provides options for feature representation

Raw Field Extraction

- I was most comfortable with Yang's statistical representation of subflows
    - Time-related (subflow duration, packets/sec, bytes/sec)
    - Packet size stats (mean, std, min, max, q1, q2, q3)
- I revisited the PCAP XML file and picked fields necessary to obtain these statistics
    - This was done through inference
    - frame.time_epoch
        - This is the packet's capture time (or arrival time)
        - Needed for subflow durations
        - Time since Unix epoch (Jan 1, 1970)
        - A numpy function converts time epochs to seconds as necessary

    - ip.len
        - Packet size in bytes
        - Specifically: Total length of IP frame
- For partitioning, Yang's study uses the following 5-tuple
    - (ip.src, srcport, ip.dst, dstport, ip.proto)
    - This means we must extract ports for both TCP and UDP
        - tcp.srcport, tcp.dstport
        - udp.srcport, udp.dstport
- Additionally, Yang's study notes the usefulness of TCP flags and IP TTL values
    - ip.ttl
    - tcp.flags
- I used tshark to extract the fields and save to CSV
    - tshark -r file.pcap -E header=y -E separator=, -E quote=d -E occurrence=f -T fields -e frame.time_epoch -e ip.len -e ip.proto -e ip.src -e ip.dst -e ip.ttl -e tcp.srcport -e tcp.dstport -e tcp.flags -e udp.srcport -e udp.dstport > file.csv

Combining Ashley's PCAPS

- Once I had the command for extraction, I needed to run it on all of Ashley's PCAPS
- I decided to combine all of the CSV files into a single CSV
    - I could have loaded all the CSV files up individually and concatenated them in pandas
    - However, I found it more straightforward to combine all the CSVs beforehand
- Preparation
    - All of Ashley's PCAPS were moved into a single folder
    - I renamed the files to make them easier to process with a shell script
        - pcap_one: pw1_1 . . . pw1_5
        - pcap_two: pw2_1
        - pcap_tree: pw3_1 . . . pw3_12
- Shell Script
    - After the renamed PCAPS were placed in a single folder, I created a shell script to run the tshark command on all of them
    - The script combines all CSV files for each group into one using Linux's cat command
        - cat pw1*.csv > combined_pw1.csv
        - cat pw3*.csv > combined_pw3.csv
    - The three CSV files are then concatenated into one large file
        - cat combined*.csv > combined_5G_pcaps.csv
    - All other csv files are removed

Data Preprocessing

- Yang's study is concerned only with IP packets, so I dropped all non-IP packets by checking for NaN values in the ip.proto column
- Remaining NaN values are converted to 0
    - ICMP packets do not have port numbers. A 0 should be a sufficient placeholder because it is part of the reserved port numbers and would not normally be used
    - Packets using UDP will have their TCP flags to 0, indicating no TCP flags

- The srcport and dstport columns are set to the corresponding TCP or UDP port numbers
    - This is done by taking the max of the TCP and UDP port
    - The original source and destination port columns (tcp.srcport, tcp.dstport, udp.srcport, udp.dstport) are dropped
- Since every CSV file has a header row, the concatenation of CSV files results in several header rows. All are dropped except the first one.
- All columns are then converted to an appropriate type
    - Strings for ID columns, datetime for time epochs, and integers for the rest
    - This is done because every column uses the raw "object" data type when first loaded

Flows

- Although Yang used the entire 5-tuple to partition the packets, I selected the tuple (ip.dst, ip.proto) for my experiments
    - I felt partitioning only by destination would better emulate "flows" into a device from multiple sources. This should help detect DDoS attacks
    - The destination port is removed so a multi-port attack can be detected, but this may not be necessary
- The flows are partitioned using Pandas functions
    - Unique Flow IDs are used to select matching rows (packets)
        - All packets matching the flow ID are set aside (partitioned)
    - A dictionary is used to link the flow IDs to their corresponding list of packets (partitions)
    - There must be at least 2 packets for flow analysis, otherwise the flow is dropped
    - See Supplemental.pdf for more info

Subflows

- Partitioning the data into long flows is not enough to accurately capture average network behavior
    - It also results in too few samples for machine learning
- The flows must be partitioned into smaller "subflows" using some method
- Yang's study allows for subflow partitioning using either fixed time intervals (<= N seconds) or a timeout interval (inter-arrival time > N seconds)
    - I created both methods for experimentation
    - I found that neither method resulted in significantly more samples, so I stuck with timeout interval because Yang used it for the statistical features portion of his study
- I iterated through the partitions (flows) and used Pandas functions to locate indices for subflow partitions
    - For timeout interval subflow selection, the primary functions are diff() and loc()
    - The supplemental PDF has more details on this process
- Note that in a live system, there would be no partitioning into "subflows"
    - We'd capture the last N seconds of traffic, partition it into flows, and extract statistical features
    - The autoencoder would tell us if the current network flow stats are normal or anomalous

Baseline Statistical Feature Selection

- As noted under the "raw field extraction" section, Yang's study used 10 statistical features as a baseline
- I dropped subflow duration from the feature list
    - I don't think network statistics need to be tied to specific durations
    - In a live system, there will only be a small window of network traffic analyzed at any time, perhaps just a few seconds
    - The stats during a 5 second capture may be normal, but if the autoencoder is trained to associate it with a 30 second flow, it will mark it as anomalous
    - What is important is that the stats are within our "normal" range, regardless of whether we analyze the last 5, 10, or 30 seconds in the real-world system

Additional Statistical Feature Selection

- Yang also noted that header information like TCP flags and IP time-to-live (TTL) can be useful for anomaly detection
    - Including information about ACK and SYN flags can help detect attacks like a SYN Flood
    - Time-to-live may be tied network topology, so differences here can indicate traffic from new devices.
    - Yang notes this may not be a useful feature if the TTL values are the same for all devices
- Yang appears to have tacked on the distribution of header values for each packet in a subflow to the end of his statistical feature vector
    - This results in a very long feature vector!
    - We'd also have to deal with uneven feature vectors because each subflow has a different number of packets
        - Yang discusses this in parts of the paper
- For ease, I chose the following statistical representations
    - ACK/sec
    - SYN/sec
    - Average TTL

Feature Extraction

- Pandas functions extract most of the features from the subflow partitions (which are Pandas dataframes)
    - The number of packets in a subflow is obtained using shape[0]
    - The duration is the difference from the last time epoch to the first
    - If there is only 1 packet in the subflow, it is skipped (same idea as before)
    - If the subflow duration is less than 1, it gets set equal to 1 for division purposes
- For ACK/sec and SYN/sec, I used bitmasks to isolate the individual flags before using Pandas sum() function to find the count
- Note that I changed the bytes/sec feature to MBits/sec because it is the standard representation of network traffic per second

Extractor Options

- As noted in the subflow section, my tool allows for partitioning by fixed interval or timeout period
    - Fixed interval is a better option when the flows do not have many breaks in them
    - Constant flows can't be broken up using a timeout period
- I also allow for the MBits/sec feature to be changed to KBits/sec
    - This was due to my Modbus experimentation
    - Network speeds were not high enough to justify using MBits
    - This is irrelevant to the 5G study
- Malicious subflows
    - These synthetic subflows are meant only for 5G testing
    - Based on ICMP ping flood
    - I used the minimum packet size (64 bytes)
    - The autoencoder does a good job of detecting these due to high packets per second
    - See the supplemental PDF for more info on malicious subflow generation

Autoencoder

- The autoencoder is a cross between Preston's autoencoder and Yang's autoencoder
- I copied Yang's default autoencoder structure
    - input layer d
    - hidden layer d-1
    - latent layer ⌈d/2⌉
    - hidden layer d-1
    - output layer d
- Activations
    - Although Yang used LeakyReLU for his autoencoder, I ran some tests with Preston's original activations (ReLU, with Linear for output layer)
    - Preston's activations showed a better grouping of normal subflows than Yang's
    - I feel that we can more easily identify a clear threshold with these activations
- Optimizations
    - I attempted to implement some of the optimization ideas from the source I listed at the end of this paper
        - Tied Weights
        - Orthogonal weights
        - Unit norm constraint
    - No individual method or combination of these resulted in improved performance
        - I will have to rerun these experiments to provide specific details on the results of each
- Threshold
    - The remainder of the autoencoder reuses Preston's code
    - The threshold can be adjusted to improve the false positive rate, although it is already apparently low

Sources

https://github.com/chicago-cdac/netml (netml tool)

https://arxiv.org/pdf/2006.16993 (Yang)

https://arxiv.org/pdf/2004.13006

- Unused but relevant
- Different "NetML": Not actually the netml tool, but a dataset
    - They used an "Intel proprietary flow feature extraction tool"
- Discusses statistical flow features and other possible features
- Does not discuss Autoencoders explicitly, but mentions other machine learning algorithms including neural networks

https://www.wireshark.org/docs/dfref/ (Wireshark reference page)

https://towardsdatascience.com/build-the-right-autoencoder-tune-and-optimize-using-pca-principles-part-ii-24b9cca69bd6 (Autoencoder Optimization)