Field Extraction (PCAPS)

Selection

To replicate the study, I needed to extract the following fields from each packet:

- frame.time_epoch
- ip.len
- ip.proto
- ip.src
- ip.dst
- tcp.srcport
- tcp.dstport
- udp.srcport
- udp.dstport

For header information, the following fields can also be used:

- ip.ttl
- tcp.flags

Extraction

The fields can be selected and saved to a CSV using Wireshark. I found it easiest to use tshark commands in shell scripts. The individual shell scripts vary depending on requirements (such as concatenation of CSV files). The primary command is:

```
tshark -r some_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 > some_file.csv
```

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.

Flows

The main idea is to group packets together based on some chosen identifiers. The study used a 5-tuple to define 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).

Partitioning

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

Dictionary

A dictionary is created to link IDs to flows (partitions). The ID dataframe is converted into a list of identifiers, and each identifier is converted to a string for use as a key in the dictionary. The associated partition becomes its value only if it contains at least 2 packets (the minimum for flow analysis).

Subflows

Two methods are available for partitioning flows into subflows: timeout period and fixed time interval.

Timeout Interval

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. Note that inter-arrival time is the moment of packet capture.

The saved indices are iterated through and, using Pandas' loc() function, sub-partitions are saved as subflows. Note that since the saved indices are the beginnings of new subflows, the last index is excluded for all subflows (except the final one). 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 dataframe (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)

I think the code is really intuitive here, so I won't go too much into it. All feature extraction is done using Pandas operations on the subflow partitions. Subflows are ignored if they have fewer than 2 packets. The study mentions a few different ways to represent subflow features, and this is just the method I felt comfortable with. I am open to experimenting with other representations.

The baseline features I used are mbits/sec, packets/sec, and several statistics on packet sizes: mean, standard deviation, quartiles (q1, q2, q3), min, and max. I don't think stats should be linked to specific durations, and I added some statistical header information (SYN/sec, TTL average) to hopefully improve anomaly detection.

Malicious Subflows

As a starting point, I created a function that generates malicious subflows. A number of malicious subflows is selected, and a loop runs that repeatedly calls another function mal_subflow() which returns a malicious subflow. The malicious subflows are saved as a dataframe and concatenated with the nominal subflows. A 'Malicious' column identifies the type of subflow. I chose 20% of the total number of subflows as my number of malicious subflows.

Synthetic Ping Flood

To simulate a basic ping flood, I wanted a high number of packets per second (pps). Since the highest pps in Ashley's subflows is a bit over 5000, I wanted my pps to be at least 6000. I arbitrarily selected 5 seconds as the duration with a number of packets between 30,000 and 40,000. Packets are the same size (64 bytes – the minimum). Statistics are generated using these input values. Note that ICMP pings do not have any port information, and an arbitrary TTL was selected.

More Synthetic Attacks

I want to modify the generate_mal_subflows() function so that we can choose the type of attack. The next attacks I want to generate are SYN flood and Query Flood since those are the example attacks I have in the MODBUS PCAPS. They should be fairly easy to replicate, and I know that we may replace them with Scapy attacks soon.

MODBUS

The introduction of the MODBUS PCAPS to my study required me to add some options to the extractor. We can set it to kilobits rather than megabits, and we can turn off malicious subflow generation since it's meant only for the 5G subflows at the moment.

Note: The "clean" MODBUS PCAP eth2dump-clean-1h_1.pcap is a duplicate. The data is already in eth2dump-clean-6h_1.pcap.

Conclusion

Finally, the subflow dataframe is shuffled and saved to a CSV. That CSV file can then be loaded up and trained by the autoencoder. I am attempting to replicate and optimize the autoencoder from the original study, but we can play around with variations (including CNNs).

Links

Feature Extraction Study: https://arxiv.org/pdf/2006.16993

MODBUS PCAPS: https://github.com/tjcruz-dei/ICS_PCAPS/releases/tag/MODBUSTCP%231