Model based semantic compression of ICS network logs

a project carried out by **Dóra Neubrandt** and supervised by **Levente Buttyán**

Motivation and goal

Industrial Control Systems (ICS) become the target of several cyber-attacks. In a report created by the IBM Managed Security Services Threat Research group in 2015, cyber attacks on Industrial Control Systems are analyzed and an increasing number of attacks against these systems is observed. Preventing such attacks is difficult and cannot be done with 100% precision. Therefore, it is important to be prepared for incidents, and efficiently handle them. The success of incident response depends heavily on means to collect and analyze attack traces. Those can be extracted from various log files created continuously during the operation of the system. It is, therefore, desirable to create logs for later analysis, including network traffic captures. However, such log files and full packet captures can be large and require a lot of storage space. Our goal in this project was to develop a compression algorithm that takes advantage of the regular patterns in ICS network traffic for achieving a good compression rate.

Related work

We reviewed and compared some of the existing works in the area of semantic compression in search of the best applicable one for our task. There is one especially outstanding method both in terms of compression ratios and available resources, which is called Squish.

Squish [1] was published in 2016 so it is a new work. It uses Bayesian networks and Arithmetic coding, which is a state of the art adaptive compression algorithm. It was constructed for relational dataset compression but it is applicable for our task as packet captures could be viewed as a table. The columns of the table could be matched with the fields of the protocols used in the packet and each row of the table could represent a packet. Squish could work as a lossy and a lossless compression algorithm as well by setting an error threshold, but in our case we only interested in the lossless scenario. In certain scenarios it can achieve near-entropy compression rate. Furthermore, the authors made a publicly available C++ implementation of the method and published a technical report of the detailed working of it.

Other methods that we reviewed include: ItCompress [2] and IPzip [3]. It seems that Squish outperforms both of these.

Application of Squish for ICS log compression

We decided to use Squish for the compression of ICS network logs. Squish originally was designed to compress relational datasets but a capture file can be easily converted into a relational table, where the fields of the different protocol messages correspond to the columns and the packets correspond to rows.

Preprocessing

The authors of Squish made a publicly available C++ implementation (on github) of their method for which they provided examples as well. The project is configured as a library, which can be used to create compression programs for any relational file format. The provided example was written to compress CSV-style files.

In our case, it was straightforward that we can convert the pcap files into CSV files, and this way we can use the provided program. For this preprocessing, first, the pcap files were converted to JSON files which can be done with Wireshark. This step is needed because the built in conversion from pcap to CSV in Wireshark does not keep all the necessary information for us. Then we wrote a preprocessing Python program which converts the JSON files into CSV files. Here we had to take into consideration that the different types of messages of the used protocol in the network (in our case STEP7) have different fields, so in their corresponding tables they will have different number of columns. Thus in the preprocessing Python program we sort the messages according to their types and generate a separate CSV file for each type.

From outer headers of captured network messages, we only keep the most necessary information: logging time, source IP address, destination IP address, source port, destination port. For inner protocols TPKT, COTP and STEP7, we keep all the header fields, however, the payload of the STEP7 messages is stored as a simple byte stream for which we have another technique to store it efficiently which we will describe below.

The overall workflow of the preprocessing is the following. First we continuously capture the network traffic (e.g. with Wireshark) and in given periods of time we save the captured traffic as pcap file. Then we generate a JSON file from the pcap, and preprocess the JSON file with our Python program which outputs CSV files according to the packet types in the traffic. Then we can apply Squish to the CSVs which outputs our compressed data files.

Additional dictionary

We store the payload of the STEP7 protocol as a simple byte stream. This payload is usually corresponds to the value of a variable that the systems asks periodically, and which can be the same for a long time. It would be inefficient to store this whole byte stream in each record thus we made a separate table (CSV file) which we use as a dictionary. We store each byte streams once in the dictionary and we assign a unique number to it. Then in the other tables it is enough to store only the corresponding number of the payload data.

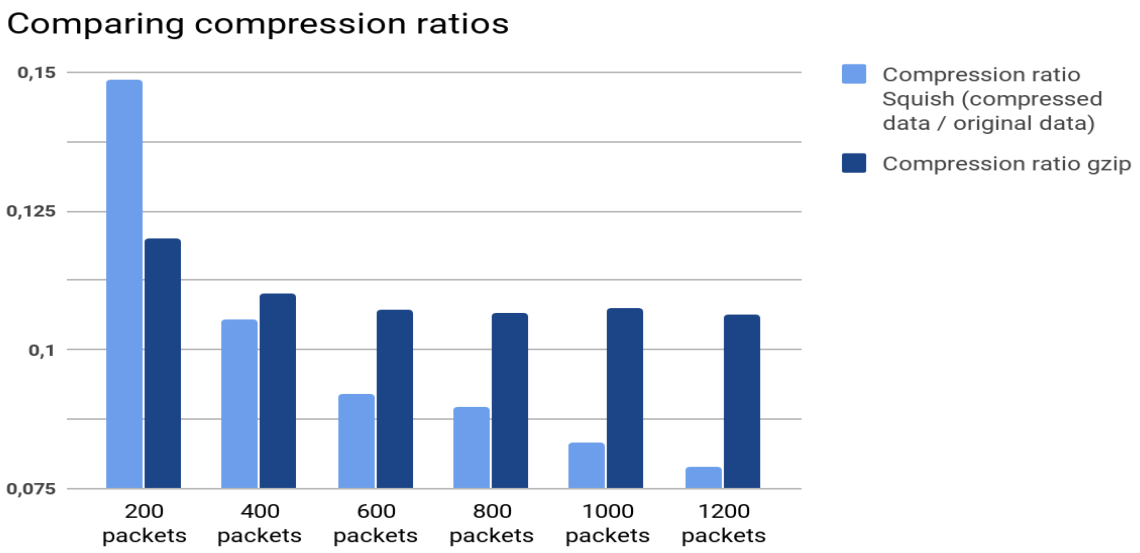
Evaluation

For testing we used the recorded network traffic from our test bed network in the CrySyS Lab. It is important to note that it does not contain all the possible types of messages of the S7 communication protocol. We plan a more comprehensive testing on more varied data in our future work.

We have chosen gzip as our baseline compression method because it is a widely and easily used compression method and the authors of the Squish also compared their solution to it.

We applied the compression methods on different data sizes. We captured the traffic of our network for a long time, and we splitted this capture file to capture files with increasing sizes. Then we applied both Squish and gzip to these files.

The figure below shows the result that we obtained:



We can see that the compression ratio of Squish is decreasing as it gets bigger files, however the compression ratio of gzip only improved in the beginning and then it remained the same. First, if we have a few packets, gzip performs better because in Squish the size of the model it built is comparable with the size of the data to be compressed, so at this size it is not efficient. Furthermore, Squish can learn more from more data, but from a point it does not learn any more, the data does not contain any further information. This we can see on the next figures where after a continuous decrease, Squish converges to a value.

Conclusions

In this project our goal was to apply a semantic compression method for the task of the compression of ICS network logs. For this we chose Squish, which is a recently published work with promising results. In order to use Squish on our data we did the necessary preprocessing steps. Furthermore, we added an additional dictionary technique to decrease redundancy thus decrease the size of the compressed data. We evaluated the performance of Squish on our captured data and we compared it with gzip. We got promising result as Squish achieved nearly 40 % better compression ratios than gzip.

References

[1] Gao, Y., Parameswaran, A. Squish: Near-optimal compression for archival of relational datasets. Technical Report. http://arxiv.org/abs/1602.04256

[2] S. Chen, S. Ranjan, A. Nucci, IPzip: A Stream-Aware IP Compression Algorithm, Data Compression Conference, 2008. DCC 2008

[3] H. V. Jagadish, R. T. Ng, B. C. Ooi, A. K. H. Tung, ItCompress: An Iterative Semantic Compression Algorithm, Proceedings of the 20th International Conference on Data Engineering, 2004.