CS230

DEEP HONEYNET ANALYSIS FOR NETWORK DETECTION

Napoleon C. Paxton
Department of Computer Science
Stanford University
ncpaxton@stanford.edu

Berkay Polat
Department of Computer Science
Stanford University
berkayp@stanford.edu

Abstract

An overwhelming majority of cyber attacks follow a sequence beginning with reconnaissance activities and ending with a total infiltration of a targeted system. Each step is recorded in network traffic that returns a level of information the attacker can use in order to be successful in the next step. In this project we explore using Recurrent Neural Networks and honeypots to train a model that can learn from the observed steps attackers are using. Successful implementation of this approach can lead to a more proactive cyber defense, which would significantly improve the security posture of a protected network.

1 Introduction and Background

Attacks on computer networks which connect to the Internet continue to rise each year. Most methods to protect against these attacks are reactionary and are constructed to protect a wide scope of threats. In this report, we explore using recurrent neural networks (RNNs) as a way to predict traffic related to attacks and we explore a tool called a honeypot to reduce the threat landscape of a computer network. We chose RNNs because they are able to use information learned from previous sequences to make predictions. This is ideal in our case since network attacks often follow a sequence of events.

1.1 Attack Sequences

The initial step in a cyber attack is reconnaissance. In this step an attacker probes a targeted network to determine if it has any weaknesses that can be exploited. Once a weakness is found, the attacker goes through a series of tests to see what tools are needed, next the infiltration takes place, then escalation of privileges, then finally the real purpose of the attack can be carried out. This purpose could be espionage, identity theft, theft of intellectual property, or even destruction of the infiltrated system. As each step is carried out, detailed metadata is written to file in the form of network traces.

1.2 Hidden Attack Traces

Network traces record all activity into and out of a network. Because of this, the full process of a cyber attacker is often captured. Unfortunately, this process from reconnaissance to total infiltration is difficult to interpret because a large majority of the traffic is considered normal and is not flagged by most status quo security tools. A major reason for this is because status quo tools attempt to protect systems from all or nearly all threats, including threats not related to the system they are trying to protect. Narrowing the scope of security threats would allow protection systems to focus solely on relevant issues.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

1.3 Honeypot Analysis

A honeypot is a security tool which includes at least one emulated service with the intent of attracting attackers to it. The goal is to capture network traces as the attackers go through their approach to compromising the system. Networks that have highly desirable services could emulate them using honeypots. If techniques exist that could learn the steps discovered with the honeypot, these techniques should be able to predict the attack before it reached the infiltration stage.

1.4 RNN Based Honeynet Analysis

A bi-product of a honeypot is that the services are not broadcast to the Internet. This means that no one is performing actions on the honeypot so the majority of activity requires a would be attacker to initiate the communication and the resulting communication traffic is considered malicious or at least suspicious. This narrows the the threat landscape to attacks to that particular service. In our approach we treat each communication as a sequence of events that can be modeled using an RNN. We believe that RNNs have the ability to learn each step of a cyber attack and this project is the start of investigating this belief.

1.5 Our Approach

In this project we attempt to classify three types of network traffic, Normal, IoT Honeypot, and Web Application Honeypot. We test three models of recurrent neural network; LSTM, Bidirectional LSTM, and GRU. The raw dataset is captured in a file format called packet captures (PCAPS). This structure is not suitable as input into our neural network, so we converted it to comma separated value (CSV) format. The labels in these datasets are generated based on the source of the type of network data.

2 Related work

In this section we discuss prior use of RNNs to predict network attacks and we also discuss honeypots.

2.1 RNNs for Network Attack Detection

There are several projects that use RNNs to classify network attacks. Many projects use a popular dataset called the DARPA Intrusion Detection Evaluation dataset. This dataset came from the 1999 Knowledge Discovery and Data Mining Tools Competition (KDD-99). Although this dataset is dated, many researchers still use it as a benchmark to test how well their algorithms detect the attacks. In Staudemeyer15, the author showed that an LSTM approach greatly exceeded the performance of all the winners of the competition, none of which used a neural network to implement their intrusion detection algorithms. Results like this and others give us more intuition that a neural network based approach detecting network attacks is feasible. Although useful, the KDD-99 dataset has been criticized over the years for not accurately depicting real-world network traffic. Because of this, a new dataset was created in 2009 called the NSL-KDD which addressed some of the shortcomings in the KDD-99 dataset. In Dhanabal18, the authors used this dataset to test their intrusion detection algorithms. They also did an analysis between protocols and attacks in that dataset, which showed a strong correlation. This gives more credence to our thought that with proper embedding, network traffic can be used to predict attacks. The approach in Elsherif18 did not use either KDD datasets but they used the same RNN algorithms we did and their datasets appear to be similar. In this paper the authors discovered that the Bidirectional RNN performed the best. This was not the case in our approach. We believe the reason is because of the in-balance of data classes. In follow up work we will ensure the dataset is balanced before running it though the classifiers.

2.2 Honeynet Analysis

There are many honeypot papers discussing the benefits of emulation of services. One such paper which is related to our approach is Britton18. Our approach to creating honeypots is similar to this approach. The authors were able to identify the likely types of attacks that would be targeting that system and then created honeypots to attract would be attackers. Their results add to our intuition

that honeypots can reduce our threat scope and provide a way to detect targeted attacker sequences. Our approach differs from theirs because they did not use neural networks to predict the classification of the traffic.

We believe all these approaches give us confidence that we are on the right track. We also believe that a major difference in our approach is that we intend to reduce the threat landscape by focusing on threats inherent to the systems we are protecting. In this way we believe our approach will be better equipped to handle new threats.

3 Dataset and Features

We created our own dataset for this project. To accomplish this, we built three virtual machines representing the type of data to be collected. Each VM contained python code to analyze PCAP data and generate a CSV file that includes 19 fields and a category in the last column. As mentioned earlier, the categories are: IoT Honeypot, Web Application Honeypot, and Normal. The IoT Honeypot is called SNARE. This honeypot uses the Telnet service to attract would be attackers. The Web application honeypot is a program called Glastoph. Glastoph emulates services related to web servers. Our normal dataset contained traffic that was assumed good for this project. In other words, we did not create a honeypot in our normal VM to attract attack traffic. We generated approximately 30 thousand examples and used an 80/10/10 split for our training, development, and testing distributions. It is important to note that each example generated by the VM is labeled based on the VM traffic it was captured on. This is important because in conventional security tools, when malicious content is found in a dataset, that content is flagged malicious but other traffic in the capture file is typically not considered malicious. Our reason for labeling all the data in the capture file as malicious or not is because non malicious traffic plays an important role in the attack process. Using honeypots to attract the network traffic also aids in this determination. Since honeypots do not present any services a user should be attracted to, all network traffic is considered malicious.

3.1 Feature Engineering

After generating the CSV files from the raw PCAP data, we needed to take of the pre-processing of the network data. Our feature space was a combination of numerical and categorical values. For categorical features, besides the IP address, we applied one-hot vectorization. For our numerical features, we applied feature normalization and scaling to overcome the range inconsistencies that could potentially hurt the overall performance of our models. These feature values ranged from having percentage values to number of bytes and therefore the final input to the models needed few adjustments. One of our features included the IP addresses where the packets were originated from. which required a special consideration. In literature, there are several ways to represent IP addresses before feeding it into a ML/DL algorithm. In order to reduce the sparsity while still being able to somewhat capture the underlying properties of each sub part of an IP address, we decided to represent them as 32-bit numbers, which had a similar structure to one-hot vectorization.

4 Methods

We experimented with three algorithms for this project, LSTM, Bidirectional LSTM, and GRU. Our main goal was to classify incoming network packets based on their origins. Mainly, we wanted to classify if a packet was from normal, web (malicious), or IoT (malicious) traffic. In order to compare the 3 models on the same benchmark, the architecture followed a similar pattern for each model. Replacing the “RNN CELL” with each of the 3 cells, we ran our models for 200 epochs. Figure 3 shows the model structure. Our hyper parameter adjustments were mainly focused on the number of time steps required for the RNN models, and the hidden layer cell sizes for both RNN and Fully-Connected Layers. The internal activation functions were chosen to be "ReLu" activations and our hyper parameter tuning experiments also included the Dropout Layer for reducing the regularization. After few experiments, a dropout value of 0.5 was chosen to be a reasonably good threshold.

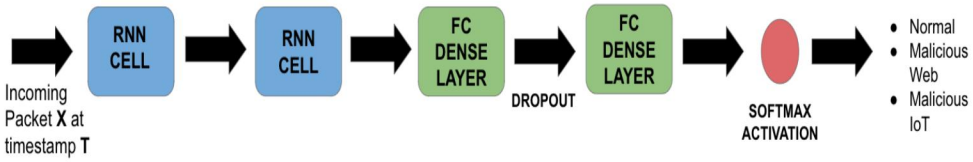

Figure 1: Model Structure

5 Experiments/Results/Discussion

5.1 Experiments and Results

Our first approach was to feed in all the network packets after sorting them out by their timestamp in order to capture the relative time difference and the order of each packet. However, the resulting algorithms were not able to capture the distinction among 3 types of traffic. This is mainly due to the heterogeneous time difference when each packet was captured. The resulting loss values over 100 epoch can be seen in Figure 2.

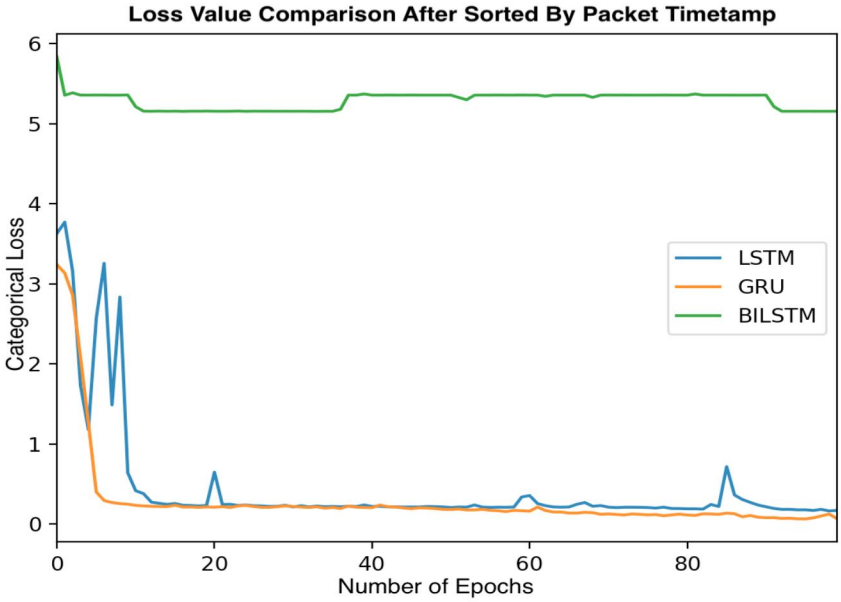

Figure 2: Model Comparison

On our second approach, we experimented with randomly shuffling all the network packets in order to overcome the high bias originated both from heterogeneous time captures and highly disproportional normal traffic to web/IoT traffic ratio. We obtained the resulting loss value characteristics in Figure 3 after running our models for 200 epochs. All 3 models still had a hard time trying to capture the complexity but it overcame the Bi LSTM anomaly.

Our results are as follows:

5.2 Discussion

This project was very interesting and challenging. We generated our own data, and we had some initial difficulty operating the data generating tools and determining how much data to generate. After running the baseline model, we realized we needed to generate more data, which really improved our results. Initially we also discussed creating an embedded matrix of some of our features, but due

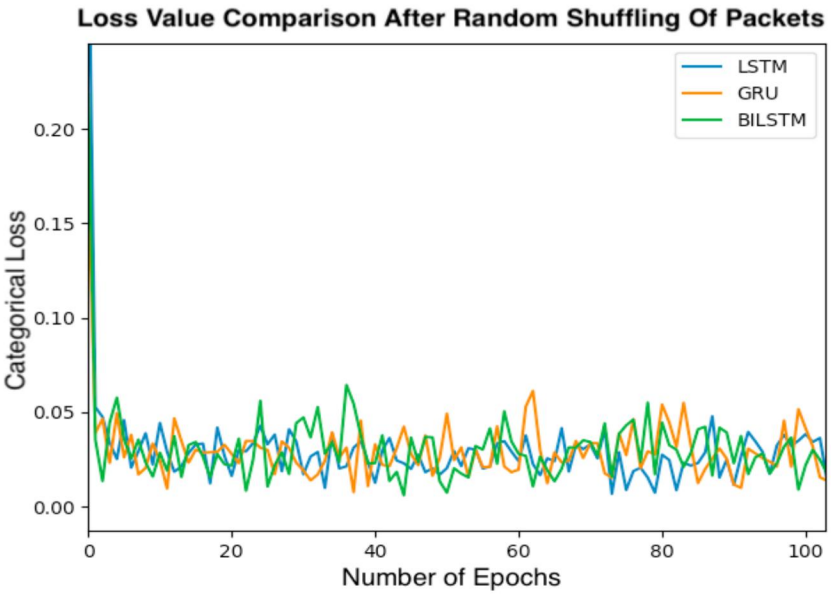

Figure 3: Loss Comparison

MODEL TYPE	ACCURACY ON THE TEST SET
LSTM	%52
GRU	%52
BILSTM	%60

Figure 4: Accuracy Results

to time decided to use one hot encoding and 32-bit encoding for our features. Overall, we believe this is a good approach to predict traffic from attackers, but more work needs to be done in terms of capturing a more meaningful relationship on packets and possibly building more complex RNN architectures to understand the complexity of multiple network features.

6 Conclusion/Future Work

Overall we were encouraged by our final results after tuning. In order to really obtain acceptable results we will need to reduce the bias. For follow up work, training a larger model will be the next step. Creating a NET2VEC embedded matrix for computer networks is also an interesting idea that would benefit this project. Providing context to fields like the nocturnal patterns of detected IP addresses could help to determine user patterns and could improve the accuracy of the classifications. More sophisticated honeypots that would be able to capture more homogeneous network traffic could also help to develop the idea of using this type of method as a protection mechanism.

7 Contributions

This project was a fully collaborative effort between Napoleon Paxton and Berkay Polat. Both students were able to communicate often, including one in-person meeting. Napoleon was the cyber

security subject matter expert, so he was the major contributor to the data creation and collection. His tasks included setting up the honeypots, as well as generating and capturing the data in a raw CSV format. Berkay Polat led and performed most of the programming in this project. His tasks included creating the RNN models as well as writing the programs to pre-process the raw data. Aside from leading their tasks, both Napoleon and Berkay contributed and supported all aspects of the project.

8 Code

Most of our work is hosted in a GitHub repository. It can be accessed via this link: https://github.com/ncpaxton/Deep-Honeynet-Analysis.git

References

[1] Staudemeyer, R.C.(2015) Applying long short-term memory recurrent neural networks to intrusion detection. In*SACJ No.56*, DOI:10.18489/sacj.v56i1.248

[2] Dhanabal, L., and S. P. Shantharajah. "A study on NSL-KDD dataset for intrusion detection system based on classification algorithms." *International Journal of Advanced Research in Computer and Communication Engineering 4.6 (2015): 446-452.*

[3] Elsherif, Ahmed. "Automatic Intrusion Detection System Using Deep Recurrent Neural Network Paradigm." (2018).

[4] Britton, T., Liu-Johnston, I., Cugnière, I., Gupta, S., Rodriguez, D., Barbier, J., & Tricaud, S. Analysis of 24 Hours Internet Attacks.

[5] Babu, Mohammed & R, Vinayakumar & Kp, Soman. (2018). RNNSecureNet: Recurrent neural networks for Cybersecurity use-cases.10.13140/RG.2.2.21876.81283.

[6] Abien Fred M. Agarap (2019). A Neural Network Architecture Combining Gated Recurrent Unit (GRU) and Support Vector Machines (SVM) for Intrustion Detection in Network Traffic Data. arXiv:1709.03082v8

[7] Zhang, Hongpo & Wu, Chase & Gao, Shan & Wang, Zongmin & Xu, Yuxiao & Liu, Yongpeng. (2018). An Effective Deep Learning Based Scheme for Network Intrusion Detection. 682-687. 10.1109/ICPR.2018.8546162.