Logo

Description automatically generated

Capstone Project – Intrusion Detection using AI

Craig Perkins – [perkins.cr@northeastern.edu](mailto:perkins.cr@northeastern.edu)

Northeastern University

January 2023 – EAI6980

Capstone Project – Intrusion Detection using AI

For the capstone project, I will be using artificial intelligence to develop an intrusion detection system. Over the last few decades, more and more companies have been relying on technology to host mission critical applications that keep industry running. In the last decade, cloud computing has emerged as an enormous field where small companies can leverage the same infrastructure as major technology companies like Microsoft, Google and Amazon. In addition to cloud services, many large companies also host applications on-premise in a data center that is managed by the same company or co-located in a data center. When securing a physical location like a data center, companies can install cameras to monitor for unauthorized entry to restricted locations to detect intrusions. It’s also important to monitor the digital infrastructure inside the data center which is connected to the internet and accessible by the outside world. There are many methods for securing servers like protecting the server with username and password, restricting traffic to certain ports and adding a firewall to block traffic. Even with all of the security measures in place, hackers can still find methods to exploit to remotely takeover machines. There have been prominent examples of zero-day vulnerabilities that allow attackers to remotely execute code on a vulnerable application server. Examples include heartbleed and log4shell.

In this project, I seek to find a solution for automated monitoring for intrusion detection to determine anomalous behavior within a cluster that should trigger a security alert. I will be examining network traffic to determine if and when a machine was compromised.

**Dataset**

For this project I will be examining raw network packets from the UNSW-NB 15 (University of New South Wales) dataset. The dataset can be located here: <https://research.unsw.edu.au/projects/unsw-nb15-dataset>. The dataset has nine types of attacks:

1. Fuzzers
2. Analysis
3. Backdoors
4. DoS
5. Exploits
6. Generic
7. Reconnaissance
8. Shellcode
9. Worms

All data and python notebooks for this project can be found in a public repository for my Github account here: <https://github.com/cwperks/EAI6980>. Below is a data dictionary of data in the dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Name** | **Type** | **Description** |
| 1 | srcip | nominal | Source IP address |
| 2 | sport | integer | Source port number |
| 3 | dstip | nominal | Destination IP address |
| 4 | dsport | integer | Destination port number |
| 5 | proto | nominal | Transaction protocol |
| 6 | state | nominal | Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state) |
| 7 | dur | Float | Record total duration |
| 8 | sbytes | Integer | Source to destination transaction bytes |
| 9 | dbytes | Integer | Destination to source transaction bytes |
| 10 | sttl | Integer | Source to destination time to live value |
| 11 | dttl | Integer | Destination to source time to live value |
| 12 | sloss | Integer | Source packets retransmitted or dropped |
| 13 | dloss | Integer | Destination packets retransmitted or dropped |
| 14 | service | nominal | http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service |
| 15 | Sload | Float | Source bits per second |
| 16 | Dload | Float | Destination bits per second |
| 17 | Spkts | integer | Source to destination packet count |
| 18 | Dpkts | integer | Destination to source packet count |
| 19 | swin | integer | Source TCP window advertisement value |
| 20 | dwin | integer | Destination TCP window advertisement value |
| 21 | stcpb | integer | Source TCP base sequence number |
| 22 | dtcpb | integer | Destination TCP base sequence number |
| 23 | smeansz | integer | Mean of the ?ow packet size transmitted by the src |
| 24 | dmeansz | integer | Mean of the ?ow packet size transmitted by the dst |
| 25 | trans\_depth | integer | Represents the pipelined depth into the connection of http request/response transaction |
| 26 | res\_bdy\_len | integer | Actual uncompressed content size of the data transferred from the server’s http service. |
| 27 | Sjit | Float | Source jitter (mSec) |
| 28 | Djit | Float | Destination jitter (mSec) |
| 29 | Stime | Timestamp | record start time |
| 30 | Ltime | Timestamp | record last time |
| 31 | Sintpkt | Float | Source interpacket arrival time (mSec) |
| 32 | Dintpkt | Float | Destination interpacket arrival time (mSec) |
| 33 | tcprtt | Float | TCP connection setup round-trip time, the sum of ’synack’ and ’ackdat’. |
| 34 | synack | Float | TCP connection setup time, the time between the SYN and the SYN\_ACK packets. |
| 35 | ackdat | Float | TCP connection setup time, the time between the SYN\_ACK and the ACK packets. |
| 36 | is\_sm\_ips\_ports | Binary | If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0 |
| 37 | ct\_state\_ttl | Integer | No. for each state (6) according to specific range of values for source/destination time to live (10) (11). |
| 38 | ct\_flw\_http\_mthd | Integer | No. of flows that has methods such as Get and Post in http service. |
| 39 | is\_ftp\_login | Binary | If the ftp session is accessed by user and password then 1 else 0. |
| 40 | ct\_ftp\_cmd | integer | No of flows that has a command in ftp session. |
| 41 | ct\_srv\_src | integer | No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26). |
| 42 | ct\_srv\_dst | integer | No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26). |
| 43 | ct\_dst\_ltm | integer | No. of connections of the same destination address (3) in 100 connections according to the last time (26). |
| 44 | ct\_src\_ ltm | integer | No. of connections of the same source address (1) in 100 connections according to the last time (26). |
| 45 | ct\_src\_dport\_ltm | integer | No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26). |
| 46 | ct\_dst\_sport\_ltm | integer | No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26). |
| 47 | ct\_dst\_src\_ltm | integer | No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26). |
| 48 | attack\_cat | nominal | The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms |
| 49 | Label | binary | 0 for normal and 1 for attack records |

The dataset is split into a train dataset with 175,341 and a test dataset with 82,332 records.

The dataset has 2 labels, 1 binary and 1 multi-classification. The binary value signifies if an entry is malicious or benign and the `attack\_cat` is the category for the attack. This analysis will build predictors for both the binary and multi-class cases.

**Exploratory Data Analysis**

Before starting to train a model, I wanted to get familiar with the dataset and understand the features contained within. As part of the exploratory data analysis (EDA), I was looking at the class balance of the dataset, looking for missing values and devising a strategy for imputation if needed, looking for opportunities to engineer new features and performing scaling or one-hot encoding on columns where needed. In the EDA, I also created charts to intuit data and challenged myself to look for patterns to see if I could predict whether a packet was malicious or not or determine the type of attack if it was malicious. Below are 5 sample records:

A picture containing table

Description automatically generated

The first figure is the number of attacks by category. In this figure you can see that normal activity is the dominant type of network activity in the dataset, the next highest category is generic malicious activity. After that, the other categories of attacks have a sizable number of records but Analysis, Backdoor, Shellcode and Worms do not have a lot of records representing the categories. The categories may need to be oversampled to augment the training set, more on class sizing is discussed in the next section on building the models.

Chart, bar chart

Description automatically generated

The next plot shows the breakdown of packets into 2 categories: normal and malicious. More than half of the dataset is composed of malicious packets, but it is evenly distributed between normal and malicious. For the training set, I think this is a good representation of both classes of data to train with.

Chart, pie chart

Description automatically generated

During the EDA I checked for missing values to determine if missing data needed to be imputed, but the dataset contains no missing values.

The next figure shows the correlation matrix of all features in the feature set. With this analysis I was looking to see if any features could be dropped from the dataset for not adding value, but this analysis revealed that each column of the dataset added unique information.

Graphical user interface

Description automatically generated

**Data Preparation for Machine Learning**

In addition to looking for missing values, I also transformed categorical data using one hot encoding and scaled numerical features using MinMax scaling.

* Categorical: proto, state, service, attack\_cat
* Binary: is\_sm\_ips\_ports, is\_ftp\_login
* Numerical: All other features

With the correlation matrix above it was found that there were 8 features with >95% correlation. These features are ['sbytes', 'dbytes', 'sloss', 'dloss', 'dwin', 'ct\_src\_dport\_ltm', 'ct\_ftp\_cmd', 'ct\_srv\_dst'] and were removed from the dataset before training as they do not add a lot of information.

After one hot encoding, the final dataframe had 180 features. At this stage the dataframe was ready for analysis with different machine learning techniques. The next step I took was to determine if any sampling was necessary to balance the classes for training. Earlier in the EDA, it was shown that the training dataset had a fair mix of benign and malicious activity so no further sampling was performed for the binary classifier.

**First Model – Binary Classifier using Logistic Regression**

For the first machine learning model, I chose to use logistic regression to create a binary classifier to predict if a packet was malicious or benign. When building the classifier I performed a train-test split on the training data to be able to perform a GridSearchCV when tuning hyperparameters. I set the test set to 25% for a total of 61,749 records in the training set and 20,583 in the test set. The first model made was using the LogisticRegression in scikit-learn and running it on the dataframe without any hyperparameter tuning. This first model yielded 91% accuracy with the following confusion matrix:

Chart

Description automatically generated

This first model sets the bar for other models to follow. 91% accuracy is higher than I had expectations of, but is not good enough for production use of the model. In production, a model with 91% accuracy would create too many security alerts for an end-user and risks fatiguing the user to the point where the software would not provide benefit.

In addition to accuracy and the confusion matrix, I also plotted the ROC curve for the model. The ROC Curve – or Receiver Operator Characteristic – shows the tradeoff of TPR and FPR for different thresholds for the classifier. In the logistic regressor, values equal to or above 0.5 would map to a 1 and values below 0.5 to 0. The ROC curve determines the LPR and FPR rates where the cutoff rates varies from 0.5.

Chart

Description automatically generated

**AutoGluon**

One of the strategies I tried to find the best model was to try an AutoML library like AutoGluon from Amazon. AutoML libraries like AutoGluon allow you to take advantage of automatic hyperparameter tuning and model selection by trying a dataset against many different machine learning models and creating a leaderboard based on the metric being optimized for. By default, accuracy is optimized. When using AutoML I built a predictor on a sample of 50,000 records from the training set and tested the predictor against the testing dataset which contains 9 million records. For the first iteration of AutoGluon, I built a binary classifier to predict whether a record in the Dataset was benign or malicious. Below is the leaderboard of models for the binary predictor:

Table

Description automatically generated

After running AutoGluon I was surprised that the logistic regression above had higher accuracy than any of the models in the leaderboard here. I decided to test the logistic regressor against the same test dataset of AutoGluon instead of using the dataset from the train\_test\_split. I noticed that with the separate testing dataset from UNSW NB15 and following the same preprocessing steps that there were some extraneous columns because of one-hot encoding and the testing dataset having values or absent of values present in the training set. When running the logistic regressor through the test dataset from another file, the accuracy was much lower at 0.44.

**Neural Networks**

AutoGluon established a new high for accuracy to beat for the neural network models. For neural networks, I built a multi-layer perceptron, an LSTM model and a GRU model using Keras to try to beat the best model from AutoGluon which had an accuracy of 90.6%. AutoGluon tried 2 Neural Networks in the model selection process: NeuralNetTorch and NeuralNetFastAI. On introspection of the predictors created by AutoGluon, it was found that the neural network architectures were generally simple with a low number of epochs and a few layers.

A picture containing timeline

Description automatically generated

**References**

1. The Heartbleed Bug - <https://heartbleed.com/>
2. Log4Shell a year on - <https://usa.kaspersky.com/blog/log4shell-still-active-2022/27531/>
3. UNSW-NB15 Dataset - <https://research.unsw.edu.au/projects/unsw-nb15-dataset>