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 managed by another company. 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 a network is 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** – In Software Development there is a testing pattern known as fuzz testing which means trying any and all inputs to a function. There is no pattern to the inputs, it is just random. Fuzzers are similar where attackers try at random to try to get information about the system they are trying to get access to.
2. **Analysis** – In an analysis attack, an attacker tries to intercept and analyze traffic to the network to glean information. A common example of this is unencrypted http traffic using Basic HTTP Authentication where a user’s password can be extracted from the HTTP Authorization header.
3. **Backdoors** – A backdoor is a way of circumventing normal Authentication and Authorization to access a system.
4. **DoS – Denial of Service** – DoS is a distributed attack where an attacker tries to take down a service by overloading it with traffic
5. **Exploits** – Exploits are known security vulnerabilities that hackers can exploit on unpatched software
6. **Generic** – Generic here is an uncategorized attack
7. **Reconnaissance** – Reconnaissance attacks are similar to analysis, but they may call a service’s APIs to gather more information about the structure of the service
8. **Shellcode** – Remote Code Execution
9. **Worms** – A warm distributes itself amongst a network

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 the 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. For binary classification, no sampling was performed. In future work expanding on multi-class classification, I plan to oversample the categories with lower representation in the training set.

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 177 features. At this stage the dataframe was ready for analysis with different machine learning techniques. 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 on the validation set with the following confusion matrix:

Chart

Description automatically generated

This first model sets the bar for other models to follow. 91% accuracy was higher than I had expected, 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. For the test set, the accuracy dropped to 85%.

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

In addition to logistic regression, I also used sklearn to try out 4 other types of models: Decision Tree Classifier, Random Forest, XGBoost and Ada Boost classifiers. Out of the 5 sklearn models, the Random Forest Classifier produced the model with the best accuracy with an accuracy on the test set of 89.2%.

A picture containing timeline

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 the training set and tested the predictor against the testing dataset. For memory intensive AutoML libraries, I used a sample of the training dataset for building the model. 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 found that the model with the highest accuracy was RandomForestGini with 90.6% accuracy. This was consistent with the sklearn results from above.

**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 used to train and a few layers.

The first table below is the model evaluation of the Neural Networks produced on the validation dataset (25% of the train dataset). The models generally all had similar accuracy, but varied greatly in the time to train.

Table, calendar

Description automatically generated

The table below is evaluating the model on the test dataset that is separate from the original train dataset. The dataset above is evaluated on the validation dataset acquired from performing a train-test split on the original dataset with 75% of the data for training and 25% for test. You can see from this table that the highest accuracy is the MLP with 3 layers, 20 nodes in each layer and relu for activation. The highest accuracy here is 98.98%.

Calendar

Description automatically generated

These tables also include the time to train and time to predict when evaluating each model. To evaluate time to train and time to predict, I used python multithreading to evaluate multiple test datasets in parallel to establish an accurate time to predict.

The depiction below shows a visualization of the accuracy and loss over the epochs trained for the 4-layer MLP. After the first 50 epochs there were diminishing returns for each subsequent epoch. The time to train the model was reasonable on a single computer, but when training time is a constraint then training can be stopped when accuracy between subsequent epochs is lower than a certain threshold or if it goes lower.

Graphical user interface, chart

Description automatically generated

Keras provides a method to plot models using pydot and graphviz. Below is a depiction of the best model that shows the layers of the network and the number of neurons in each layer. Determining the width and depth of a neural network to use is challenging, and it’s not always immediately obvious how to architect a neural network to solve a particular problem. I found this DataExchange post (<https://stats.stackexchange.com/a/223637>) helpful to build intuition when figuring out how to design a Neural Network.

Diagram

Description automatically generated

**Other AutoML Libraries**

In addition to AutoML, I also evaluated PyCaret, Autosklearn and Autokeras. Out of these 3 libraries, I found that PyCaret was the easiest to use but the most memory intensive. Autosklearn was not well documented and autokeras did not produce great models. Please refer to each respective Jupyter notebook to see each AutoML library in use. PyCaret produced the following leaderboard:

Text, table

Description automatically generated

Along with the leaderboard, I also used PyCaret to list the feature importances for the best model that it produced.

Table

Description automatically generated

**Multi-Class Classification**

For most of the analysis a binary classifier was produced for analysis. Up to this point in the write-up, all accuracies and leaderboards shown were for a binary classifier to determine if a packet was normal or malicious. In practice, it’s not only important to know if a packet is suspicious, but also to try to determine the type of attack under way. I used the same Keras models from earlier and changed the output layer to produce multi-class classifiers. The models produced had noticeably less accurate predictions than the models performing binary classification.

Calendar

Description automatically generated with medium confidence

Below is a confusion matrix produced from evaluating the 4-layer multi-class multi-layer perceptron (MLP)

Graphical user interface

Description automatically generated

**Future Work**

In this project, I focused greatly on the model building to synthesize all that I have learned over the past 2 years in CPS, but looking forward I would like to try to deploy this model in a production setting to see if it could be productized and put to use in commercial applications. In this project, I chose accuracy as the metric to pick the best model, but in future iterations I would also consider using accuracy in combination with the False Positive rate. Alert fatigue is a big problem in automated monitoring systems and security analysts could start dismissing alerts if they do not trust the software to identify suspicious patterns reliably.

I would also like to expand upon this project by adding log analytics capabilities to look for threats within application log data and consider other threats like Denial of Service. For instance, if there were logs from an enterprise’s Active Directory service, I would like to write software to identify malicious login attempts and create alerts for common exploits.

**Conclusion**

Machine Learning is a useful tool in the fight against Malware. This project demonstrated how Artificial Intelligence can be used to identify suspicious network traffic to alert a Security Analyst of the need for further investigation. Cybersecurity is a complex field and becoming ever more important as more devices are connected to the internet. Nations have critical infrastructure connected to the internet that is subject to espionage and attacks by malicious parties, so it is imperative to stay ahead of adversaries when securing the assets. This was a great project to apply my learnings from this program to a real-world problem that provides value.

Artificial Intelligence is the new electricity as Andrew Ng has famously stated. This project shows a narrow application of Artificial Intelligence trained for a specific purpose of analyzing packet metadata, but this can be expanded upon to provide a full-suite security solution for an enterprise. By combining the solution built here with log analytics and software to analyze for known malicious binaries on a computer, an enterprise would go a long way towards providing security for their digital assets. There’s a famous saying in cybersecurity that playing defense is tougher than offense, because defense requires knowing all possible attack vectors by an adversary, whereas an adversary only needs to find the single vulnerability that would permit them unrestricted access to the protected resources. With that being said, Artificial Intelligence is well positioned to provide a suitable solution for a rapidly changing landscape as the AI can be built in such a way to be adaptable and learn from events that transpire. The Security Analysts using this system would provide important feedback that the system can use to improve over time.

**References**

1. CSO Online. (2018, April 2). The Heartbleed bug: How a flaw in OpenSSL caused a security crisis. [https://www.csoonline.com/article/3223203/the-heartbleed-bug-how-a-flaw-in-openssl-caused-a-security-crisis.html](https://www.csoonline.com/article/3223203/the-heartbleed-bug-how-a-flaw-in-openssl-caused-a-security-crisis.html#:~:text=Heartbleed%20is%20a%20vulnerability%20in,Sockets%20Layer%20(SSL)%20protocols)
2. Kaspersky. (2022, January 3). Log4Shell still active in 2022. <https://usa.kaspersky.com/blog/log4shell-still-active-2022/27531/>
3. UNSW-NB15 Dataset - <https://research.unsw.edu.au/projects/unsw-nb15-dataset>
4. Shi, Z. (2019, September 20). Tabular Prediction using Auto Machine Learning (AutoGluon). Towards Data Science. https://towardsdatascience.com/tabular-prediction-using-auto-machine-learning-autogluon-de2507ecd94f
5. Kaya, Y., & Aydin, M. A. (2020). On the Interpretability of Artificial Intelligence Models in Healthcare. Algorithms, 15(8), 258. https://doi.org/10.3390/a15080258
6. Tang, Y., Xu, M., & Wang, Y. (2020). An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. Journal of Big Data, 7(1), 1-32. https://doi.org/10.1186/s40537-020-00379-6