Final Project Report for ECE514
Real time Network Intrusion Detection System

Evangelou Sotirios SEVANGELOU@UTH.GR
Kalais Konstantinos KKALAIS@UTH.GR
Chatziefremidis Eleutherios ECHATZIEF@UTH.GR

Abstract

As a final Project for the ECE417 "Machine Learning" course in the spring of 2019, our team chose to implement a Network Intrusion Detection system, leveraging the power of Machine Learning and Neural Networks. For the final project of ECE514 "Problem Solving Environment for Data Science Applications" course in winter of 2019 we decided to make the IDS usable in real life situations where each millisecond multiple packets - probably malicious - are running through network cards and affect computer systems. We needed a way to tackle this situation, so we created a pipeline in order to collect data, apply filtering and handling, create datasets, and use our model to predict on real time whether our system is under attack and which kind of attack is happening.

1. Introduction

1.1. The problem

The so-called fourth industrial revolution has changed the modern life as we knew it. Besides personal computers, the huge network we call Internet now includes, mobile phones, tablets, IOT devices and many more people have network access and knowledge that was considered arcane 20 years ago.
The possibilities and opportunities are limitless; unfortunately, so too are the risks and chances of malicious intrusions by people or other systems.
Cybersecurity is the practice of protecting systems, networks, and programs from digital attacks, and the subprocess of the identification of these attacks is called Intrusion Detection.

University of Thessaly, ECE 417 Course Final Project.

1.2. The motivation

Intrusion Detection Systems (IDS) that monitor network traffic and anomalies are called NIDS (network intrusion detection systems) and often use an approach called anomaly-based detection which detects deviations from normal traffic behaviours.
In the past years this has been achieved by 'rules' introduced by individual engineers or 'smart' systems.
Thus, we decided to leverage the power of Machine Learning and Neural Networks in order to implement a 'smart' system that will analyse packet metadata and take decisions on the security of the network it resides in. The motivation is to contribute in the field of security and especially autonomous security, and create a system that will effectively handle the network security issues for an individual user or a whole business.

1.3. Previous Research on the issue

The use of Machine Learning techniques in developing intrusion detection systems is not a new idea.Knowing that, we decided to search for papers and research on the particular issue.
We focused on the work of two particular papers, both from research conducted by the University of New Mexico.
The first, by Mahdi Zamani and Mahnush Movahedi [1] delves into the use of Machine Learning techniques in the Intrusion Detection field of study, taking a more general approach, which familiarised us with the problem we wanted to solve.
The second paper we studied, by Mukkamala, Janoski and Sung[2] takes a closer perspective on the issue, trying to implement solutions using Support Vector Machines and Neural Network on a DARPA dataset, and comparing the two methods' results.
Based on these respected professors' work, we decided to use some of these methods in our project.
For the extension of the project, the live data streaming phase, we studied the paper of the creators of the UNSW dataset we used for training and testing, Nour Moustafa and Abdelhameed Moustafa[3]. This paper gave us a significant

insight in the collection and processing of the data that led to the creation of the data we used as training and testing samples.

2. The Dataset

2.1. General Description

The dataset we used to train our Machine Learning models and test it, belongs to the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS)[4].
The creators of the dataset used the IXIA PerfectStorm tool for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours.
The main reason we chose the dataset was because it uses as features the fields contained in PCAP files. This means, that the system can be applied in real situations, using as input packets captured in the network with Tcpdump or Wireshark, turned into PCAP files and consecutively into csv format by a simple parsing script.

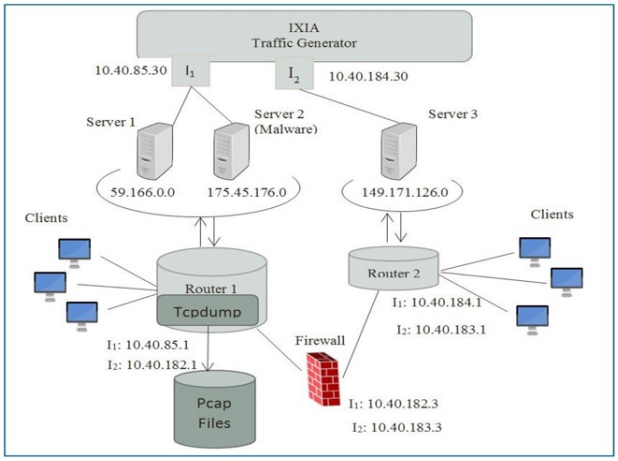

Figure 1. Testbed used to create the Dataset

2.2. Specifics of the Data

The dataset we downloaded from the UNSW website, was already split into a training set of 175341 instances and a testing set of 82332 instances. The dataset had some minor inconsistencies such as missing features that we removed during the dataset cleaning phase. The dataset consists of 43 features- fields of PCAP files including IP sources and destinations, ports, service types etc. The attack types provided as labels by the dataset (instead of normal packets) are:

- Fuzzers
- Analysis
- Backdoors
- DoS
- Exploits
- Generic
- Reconnaissance
- Shellcode
- Worms

2.3. Preprocessing

One of the main parts of Machine Learning is preprocessing, and data preparation.
In order to ensure the best performance of the algorithms we will discuss in the next section, we decided at first to find highly correlated features and drop them from the dataset. Using the FeatureSelector library by Will Koehrsen [4], we were able to drop features with at least 85% correlation, in order to make our algorithm more cost and time efficient.
Next, we use the LabelEncoder class provided by Scikit Learn library, in order to transform the nominal values to numeric and enable the algorithm to perform arithmetic operations on these values.
Lastly, we scaled our data using the MinMaxScaler class of Scikit Learn in the range of (0,1) for easier convergence. This procedure was not implemented for the SVM methodology, as SVMs are scale invariant.

3. Data Streaming Pipeline

A real time data feed to the model we will introduce below is essential. Data are being created, moved, and deleted in a split second and if we are trying to achieve ”serious” intrusion detection, we need to handle and test data the moment they pass through our network.
For us, the way to go was to create a pipeline composed of four phases/stages:

- **Phase 1:** Data collection
- **Phase 2:** Feature extraction
- **Phase 3:** Processing and Dataframe extraction
- **Phase 4:** Classification

The four phases of the pipeline can work simultaneously on different threads which means that in each cycle we can handle up to four data batches. A diagram of the pipeline is provided below:

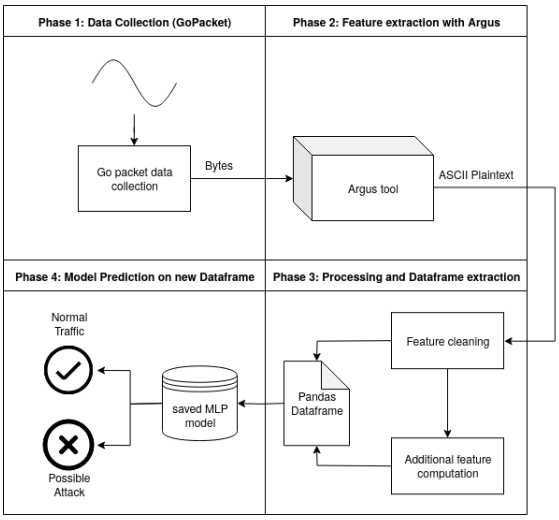

Figure 2. Pipeline overview

3.1. Real-time Data collection

Collecting data passing through the network is as easy as using the tcpdump tool to sniff the network and extract information into the form of a .pcap file. Later, using Wireshark anyone would be able to inspect the packets and extract valuable information.
However, the dataset we used for training, and therefore the dataset we need to produce from the pipeline, is much more sophisticated than parsing a .pcap file and using its fields as features.
Thus, in this stage we used gopacket, a Go lang package used for decoding network packets in order to collect packets and pass them as bytes to the next phase for inspection and processing.

```
// Use the handle as a packet source to process all packets
packetSource := gopacket.NewPacketSource(handle,handle.LinkType())
for packet := range packetSource.Packets(){
  //printPacketInfo(packet)
    w.WritePacket(packet.Metadata().CaptureInfo, packet.Data())
  packetCount++
```

Figure 3. Go code collecting packets and writing their information in the standard output

3.2. Feature Extraction with Argus

The data collected in real time need to produce the fields we are using as features for our machine learning models. Specifically, after cleaning highly correlated features, we are left with 28 features, listed in Table 1.

The light-gray colored features, are features we can extract using a very helpful tool named Argus[7]. Copying the description of the tool from its wiki, *Argus reports on the transactions that it discovers, as periodic network flow data, that is suitable for historical and near real-time processing for forensics, trending and alarm/alerting*.

Table 1. Table 1. Features used in training and testing

dur	proto	service	state
spkts	dpkts	rate	sttl
dttl	sload	dload	sinpkt
dinpkt	sjit	djit	swin
stcpb	dtcpb	tcprtt	smeansz
dmeansz	trans_depth	response_body_len	ct_srv_src
ct_state_ttl	ct_dst_ltm	is_ftp_login	ct_flw_http_mthd

Therefore, we utilised argus' reports in order to perform flow analysis into the data produced from the first pipeline phase, and extract most of the features needed for predicting an attack using a command called *ra* (read argus),.

```
4  while True:
5    os.system("argus -w -")
```

Figure 4. Using argus with stream input and stdout output

```
while True:
  os.system("ra -r - -L0 -c ,  -s -Z +dport \
            +ltime +dur +proto +state +spkts \
            +dpkts +rate +sttl +dttl +sload \
            +dload +sintpkt +dintpkt +sjit \
            +djit +swin +stcpb +dtcpb +tcprtt \
            +smeansz +dmeansz +trans")
```

Figure 5. Using ra to extract features from argus' results

After argus has analysed the input, it reports in the standard output, where ra analyses the output and produces the results - features - we need. This phases produces most of the features we need to use on our model, but some cannot be produced by argus.

3.3. Additional feature computation

Five of the features listed in Table 1 are slightly heavier coloured than the rest. These are features that contain mostly flow data and need some further processing of the data at hand to produce them.
Apart from *is_ftp_login*, which is pretty self-explainable, the other four features contain information that need more than one packet to compute. So, as the packets are passing through the pipeline and are being analysed , in the third stage they are also passing through a programming logical block that keeps state on consecutive packets satisfying a certain scenario. For example, *ct_srv_src* is keeping count on consecutive packets regarding the same source IP and service, by sequentially comparing the appropriate fields of the passing packets. Using simple Python functions we

were able to compute the last features needed to create a whole Pandas Data frame that will be forwarded to the last and most important phase.

In this phase we are also taking care to remove packets (rows in the dataframe) containing NaN fields, transform nominal fields to numeric, and scale certain fields.

Before talking about how our model predicts on the data we have prepared in real-time, one should first read the following section, regarding the machine learning and neural network models we studied, the experiments we conducted and the results they led to, and the model we decided to adopt as the top solution.

4. Algorithms and Body of Work

4.1. General Approach and Algorithms used

The dataset provides two columns as targets one with normal-not normal values, and one with attack type values. We found that classifying by attack type and aggregating the results to normal and abnormal traffic gave us better results than classifying on normal/abnormal traffic. Thus, our problem is categorised as a supervised multiclass classification problem.

Based on the papers we mentioned we decided on the use of three algorithms to focus on the issue:

- Multilayer Perceptron
- Random Forest
- Support Vector Machine

We will analyse the work implemented in each one, rated by increasing performance.

4.2. Support Vector Machine

SVMs were proposed by the above-mentioned papers as the best performing Machine Learning technique for this kind of problem.
The best parameters we used, after testing several variants with trial-and-error approach, were 100 for the C value, and 0.00001 for the gamma value. We also used an RBF kernel, which is considered the most effective in general problems. Though, our dataset was very large, and comparing to the other two algorithms the time coefficient was a factor that made it not suitable enough in our opinion. Specifically, using only about 12% of the training set, the training phase lasted about 100 seconds, and the time would increase exponentially with more data as SVM tends to cross-calculate each point with the rest of the data. Comparing to the other two algorithms the time was in a much higher scale.

The experiments we conducted with SVMs used about 50.000 training points (more points would take too much time), and produced accuracy rates of 76.101% on deciding on the normality of the network traffic (Normal/Abnormal) and 68.087% accuracy in deciding the type of attack happening in the network.

```
66 C=100
67 gamma =0.00001
68 classifier = SVC(kernel="rbf",C=C,gamma=gamma,verbose=0)
69 classifier.fit(train_x, train_y)
70 yhat = classifier.predict(test_x)
```

Figure 6. SVM using Sklearn's SVC classifier class

4.3. Random Forest

The random forest technique was not mentioned in the researched papers, but was a technique we decided to try based on the speed of execution even on large datasets.
The parameters we used after trying several variants, were 1000 estimators and 5 as the depth of the tree. The algorithm provided us with a high accuracy result in a short time, specifically 81.336% on the binary classification problem of traffic normality, and 68.087% on the attack type classification problem. Generally our opinion was that the algorithm provided us with very accurate results in regard to how simple it was to develop and fast to execute.

```
32 clf = RandomForestClassifier(n_estimators=1000, max_depth=5,random_state=None)
33 clf.fit(train_x,train_y)
34 yhat = clf.predict(test_x)
```

Figure 7. Random Forest using Sklearn's RandomForestClassifier class

4.4. Multilayer Perceptron

Using Multilayer Perceptron provided us with the best results of the three algorithms. For the neural network's architecture we used:

- A Dense input layer of 300 neurons and rectifier linear unit activation function
- A Dense hidden layer of 150 neurons and rectifier linear unit activation function.
- A Dense output layer of 10 output classes with softmax activation function.

The model was compiled with Adam optimizer and a categorical cross-entropy loss function, and trained in batches of 150 instances for 150 iterations.

The results we got were a problem-high accuracy of 83.96% on the normality classification and 82.81% on the attack type classification.

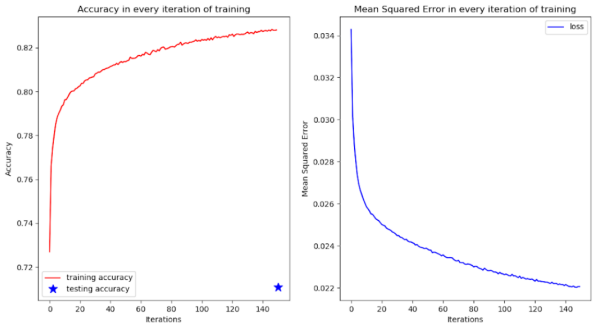

Figure 8. Plots of the Accuracy of atk type and the Mean Square Error by Iteration

For this problem it is significant that we have as less attacks classified as normal as we can. For this reason we also introduced a confusion matrix metric, that, as we can see provided as with only about 1.4% false positive classifications.

	T	F
T	24517	12483
F	1184	44148

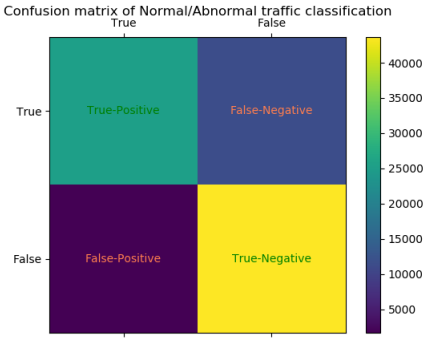

Figure 9. Confusion Matrix heat map

```
model = Sequential()
model.add(Dense(300, input_dim=trainx.shape[1], activation='relu'))
model.add(Dense(150, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['mse','accuracy'])
```

Figure 10. MLP model using Keras, Tensorflow's high-level API

4.5. Methodology Comparisons

Below, an accuracy-based comparison between the three algorithms is provided, in order to give a better look into which algorithm is deemed more suitable.

Table 2. Accuracy of every algorithm used in the NIDS project

TESTS	NORMAL/ABNORMAL CLASSIFICATION	ATTACK TYPE CLASSIFICATION
MLP	83.96%	82.81%
RF	81.33%	68.08%
SVM	76.10%	68.08%

From the table, it is obvious that Multilayer Perceptron produces the better results of the three, and we also provide a bar plot to visually confirm that MLP performed better.

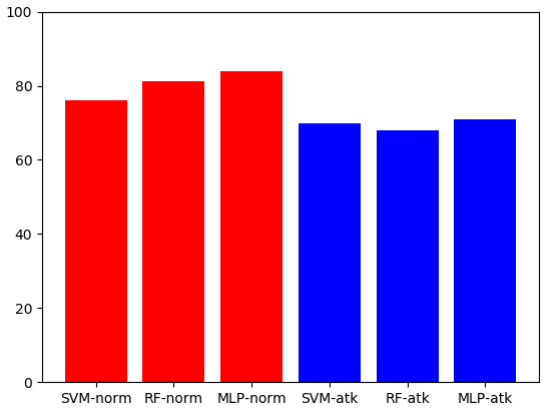

Figure 11. Bar plot for accuracy comparison between each algorithm

5. Predicting on Real Time

The final point of interest of this paper is obviously how this IDS model can predict on real life situations. In section 3 we extensively talked about a pipeline that would allow us to predict in real time data and be actively safe by detecting intrusions in the network.

After concluding that MLP produces the best results for our problem, a model and its weights were exported as separate files, allowing the reuse of the model without training every time. The dataframe (containing one row-instance per round) that was produced in the phase 3 of the pipeline is being used as input to this model after it is successfully loaded using custom Python functions. Being enclosed in an infinite while loop, the model consecutively predicts on these instances and prints the result of the prediction in the

standard output.
In the case that a possible attack is predicted, the program also informs us of the classified attack type and exports the particular instance in a csv for possible further testing that will ensure whether we do have an attack, or a false negative result.

```
Normal Behavior
Possible 'Exploits' Attack : added to out.csv for analysis.
Normal Behavior
Possible 'Fuzzers' Attack : added to out.csv for analysis.
Normal Behavior
Normal Behavior
Normal Behavior
Normal Behavior
Normal Behavior
Normal Behavior
Normal Behavior
Normal Behavior
Normal Behavior
```

Figure 12. Sample output of the real time prediction. The instances classified as attacks are added to a .csv file for more analysis.

6. Downloading and Running the project

The NIDS project is hosted on the Github software development platform, and specifically in https://github.com/EvangelouSotiris/NIDS_Project_CE417. Instructions to download dependencies, clone and run successfully the project are included in the README.md file.

7. Thanks to

The project was implemented in the spectrum of the ECE417 and ECE514 courses of the university of Thessaly, Electrical and Computer Engineering Department.
The professor and supervisor of the project, prof. E.N. Houstis guided us to the majority of the knowledge, tools and skills needed for this purpose. We would like to deeply thank him.

8. References

References

[1] Mahdi Zamani, Mahnush Movahedi *Machine Learning Techniques for Intrusion Detection*. University of New Mexico, 2013.

[2] Srinivas Mukkamala, Guadalupe Janoski, Andrew Sung. *Intrusion Detection: Support Vector Machines and Neural Networks*. New Mexico Institute of Mining and Technology, 2002.

[3] Nour Moustafa, Abdelhameed Moustafa. *Designing an online and reliable statistical anomaly detection framework for dealing with large high-speed network traffic* University of New South Wales, Australia, June 2017.

[4] UNSW-NB15 Dataset, ACCS
https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/

[5] Feature Selector library : Will Koehrsen
https://github.com/WillKoehrsen/feature-selector

[6] Gopacket package: packet decoding for Go language.
https://godoc.org/github.com/google/gopacket

[7] The argus tool.
https://www.qosient.com/argus/publications.shtml