Detection of Botnet Activity via Machine Learning

Diogo Jerónimo
diogo.jeronimo@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2018

Abstract

Botnet attacks have always been a serious problem for companies and organizations to deal with since they can engage in all sorts of malicious activity with the purpose of causing damage and stealing information from compromised machines. In this work we analyzed network trace logs from botnet connections and created a system that is capable of detecting this kind of activity leveraging a supervised machine learning approach. This created tool not only detects abnormal network behavior but it can be also used to aid forensic analysts in the detection of other related botnet attacks such as data exfiltration and DDOS attacks. We created two scenarios mixing normal and abnormal activity in the network in order to evaluate the system. In the first scenario, we tested the detector with only one botnet present, since it is the scenario that is closest to reality. In the second scenario, we mixed normal traffic with multiple types of botnets to see if it is viable for a machine learning system to detect so many patterns at once. We also use a Cross Validation strategy that respects the temporal order of the packets of each connection in the logs so that the detector does not use future information that can create bias in the classification. In the end, we present an example of a practical application of the detector with a module called Decider. The Decider chooses the machines that are most likely compromised in a network basing its decision on the results from the classifier algorithm. The solution's architecture was made to be extended in order to use information from other data sources such as file access and login logs to create clusters of information that allows the improvement of overall detection results.

Keywords: Botnets, Machine Learning, Information Security, Data Exfiltration

1. Introduction

One of the most dangerous and damaging attacks for organizations, nowadays, is the infection of machines that allow the creation of botnets. The attacker can have full access to these machines allowing the engagement of multiple damaging activities. These kinds of attacks are hard to prevent because, once a machine is infected, the attacker might issue commands to the zombie machine right away, causing immediate damage. Current defense mechanisms on the subject to detect botnets use tools like DPI to inspect the payload of packets to detect potential information carried inside the packet. If the packets are ciphered though, the tool becomes ineffective because we cannot inspect the payload anymore.

Other existing methods aim at producing mechanisms that prevent attackers breach into the systems and the network infrastructure of organizations. For example, the main focus of the detection defense mechanisms rely on tools such as IDS that alone, are simply not enough for this type of attacks. Botnets have ways to minimize their network fingerprint like communicating with the Command and Control server in random times of the day. Also, the use of obfuscation and encryption, as already mentioned, makes the task harder for the defender. Preventing the infection by malware is pretty complex too. Sometimes, the malware can be sent by email masked as a benign file or malicious insiders can have enough knowledge to plant a malware inside one of the machines, creating a botnet easily.

The work presented here will explore this topic, focusing on malware that can create botnets, more specifically, we are interested in detecting connections made by machines that were infected and are communicating with the attacker's machine. Botnets have been largely used by attackers in the past to do all kinds of malicious activities including compromising machines to monitor its activities by planting a keylogger or to sniff traffic, to launch DDOS attacks, to steal the identity of the machine, and even to exfiltrate data from the user's computer.

The main goal of the system created in this project is to detect via a supervised machine learning approach, malicious network activity by extracting generic features associated with the flows between the network machines in order to train a classifier and predict future connections. However, the system was designed to be used as an auxiliary or complementary defense mechanism to existing ones. For example, an organization logs multiple services, at various operating system levels. One could use this tool to gather information from file access and modification logs, web services logs or even login information and correlate this data with the results from our classification process to detect both botnet activity and data exfiltration.

1.1. Contribution

The method we propose has two main parts. The first one has to do with the analysis methodology and classification of botnets via Machine Learning and network trace logs. This is done by the tool that we created for this purpose. The second one has to do with a practical module called Decider that will allow us to flag the most likely infected machines using information from the classifier results.

Also, we will analyze what are the characteristics of data exfiltration since it can be useful in the future when having in mind the extension of this system to other types of attacks. We will try to answer questions like, what is data exfiltration? How can data be exfiltrated? What are the current methods that attackers use? What can we do to detect it and mitigate its damage? To do this, we will first present a survey on data exfiltration in order to better understand this problem.

2. Background

In order to get a better understanding of the problem we are dealing with, we will point out some relevant work and solutions proposed in the past. We will present relevant work in the field regarding botnet detection as well as machine learning methods to do so. Also, as an example, we will explore the issue of data exfiltration since our created tool can be used to detect this specific attack. To propose our method, we created a brief survey on previous work about these two topics. The survey will provide insight into what was developed in the area so that we can propose a method that attempts to solve the challenges faced by previous work.

Existing research on data exfiltration is very broad since there are multiple methods to do so. A data exfiltration taxonomy can be found in [3] as well as a description of some exfiltration methods. In regards to network data exfiltration detection, Liu *et al.* [6] describe a framework to be used to detect inside jobs focusing, primarily, on the outbound traffic. Inside jobs are essentially crimes committed by someone working on a company or organization sometimes being in positions of trust and power. These inside jobs can range from installing malicious software so that an external party can access information or stealing information from confidential documents. The framework is able to filter all outbound traffic and make a fine-grained identification of applications used (for example Gmail), based on some features of the traffic like temporal patterns and the size of the packets. Also, the method uses signatures of the traffic to produce fingerprints of sensitive files companies might want to protect. If such a signature is found leaving the network, the system might send an alert to the administrators.

Sometimes, getting traffic signatures is not that simple because attackers might use anti-signature techniques such as encryption. At the same time, the covertness of data exfiltration increases making the detection much more complex. He *et al.* [5] discuss methods to detect data exfiltration when the attackers encrypt the communications. The identification of encrypted packets is done by calculating entropy where high entropy values might indicate encryption. Information entropy is basically defined as how much useful information a message is expected to contain. Then, the authors try to identify data exfiltration activities by profiling the connections and leveraging a machine learning approach to classify malicious users.

In a more recent work, Nadler *et al.* [7] present ways to detect low throughput data exfiltration via DNS protocol, using a machine learning approach. Since making a lot of DNS requests to exfiltrate data would raise alarms in the intrusion detection systems of any company, sophisticated methods include making very little requests throughout the day, in order to avoid the defense mechanisms. The authors attempt to identify these stealthy communications by extracting features from logs and, use a classifier to distinguish legitimate communications from malicious ones.

A couple of physical data exfiltration techniques are discussed in [4, 8]. These authors try to address a common problem regarding the problem of the identification of malicious copying of files from a Windows filesystem in order to steal them through a physical device like a USB stick. They attempt to analyze patterns on folder copying and timestamp analysis for the detection.

In terms of botnet data exfiltration research, Al-Bataineh *et al.* [1] analyze a popular data exfiltration botnet called Zeus and propose a method to detect its activities. This work analyzes exfiltration where botnets use HTTP POST requests. We can find recent work on stealthy botnets in [10, 11] where the authors attempt to study these behaviors. The first work attempts to detect data exfiltration by deploying detectors over the network and periodically, moving them to reduce the likelihood of exfiltration. The second work uses a decision-making algorithm called reinforcement learning to deploy optimal defenses to the network such as honeypots, that detect intruders and network-based detection mechanisms that enable a defender to locate compromised machines that persist in the network. The authors aim to reduce the likelihood of the attacker to persist in the network by identifying and taking down the bots.

Work by Zhao *et al.* [12] and Beigi *et al.* [2] talk about machine learning techniques to detect botnets. In the first work, the authors base the detection in traffic behavior that allows the defender to detect malicious connections without needing to inspect the payload of packets which, can be obfuscated or encrypted. The authors make use of a Decision Tree with previously selected features from the traffic. The features are extracted in a specific time window which yield very high detection rate with very low false positive percentage. On the other hand, Beigi *et al.* extend the work by Zhao *et al.*, discussing the selection of traffic features and created a whole new dataset aiming to solve the problem of generality. In the end, the authors test the dataset with a Decision Tree algorithm.

There has been work done on data exfiltration by botnets and the stealthy behavior separately, however, there is very little research that focuses on these two aspects at the same time. That is, none of the previous research specifically studies the behavior of stealthy data exfiltration by botnets that employ anti-signature techniques like encryption and obfuscation. We believe that, with our work, this can become a reality in the near future, combining all sorts of valuable information contained in logs to extend our system.

3. Implementation

The system implemented for the thesis is a detector based on machine learning algorithms capable of identifying malicious connections engaged by botnets. The classification is supervised and because of that, we trained multiple classifiers with a publicly available dataset [2] that contains traces of more than a dozen of different types of botnets separated by a training and testing dataset. The dataset used consists of two network trace files (.pcap) each containing more than 2 GB of information about the packets exchanged that will serve as logs to be analyzed. The network traces are generated by the authors on real machines with botnets that simulate well the reality of these malicious attacks.

The system will be a useful tool not only to detect botnet activity, but to also aid analysts in identifying other types of attacks such as data exfiltration. The results from this system can provide strong correlations between logs maintained on the victim's computer such as logs that keep track of file accesses and modifications and the presence of malicious activity in the network. In the future, the detection of other threats could be explored by extending this exact architecture that we propose.

3.1. Architecture

We present here the system's architecture implemented for the solution. Previous research done on this problem discusses the main defense and monitor mechanisms but very few create fully functioning implementations capable of detecting effective botnet activity detection. We want to be able to create a prototype that will serve as a base to a detection framework supported by machine learning and that could be integrated with multiple tools such as IDS, anti-virus software, and firewalls. Figure 1 shows the architecture and it's main components as implemented in the solution. The main idea is to simulate how can this system work in the real world regarding detection of botnet activity (and potentially exfiltration) consisted of several steps beginning with the collection of log data and resulting in a full report detailing what machines are most likely to be compromised and controlled by an attacker. We want to be able to detail these reports and alerts with complete information and statistics so that forensic analysts can act upon these types of threats.

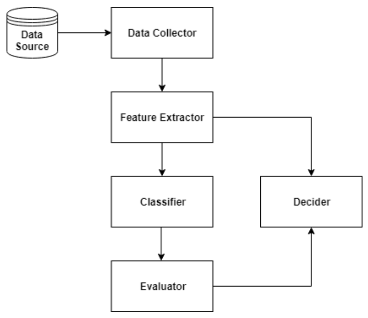

Figure 1: Diagram of the architecture

3.2. Methodology

The solution was developed in the Python[1] language version 3.6 making use of the Sckikit-Learn[2] library that implements all the machine algorithms that we need. As already mentioned, the input of our system will be the network logs from the publicly available dataset[3] found at the University of New Brunswick website in order to train and test the system. To analyze the traces we used Wire-

[1]https://www.python.org
[2]http://scikit-learn.org
[3]http://www.unb.ca/cic/datasets/botnet.html

shark[4] since it provides a nice graphical interface, and for the dataset processing, we created scripts using the Scapy[5] library in order to manipulate the data.

3.3. Analysis

Before implementing the system, we had to first analyze the network logs that we gathered from the datasets so that we could later, extract relevant information that we could use on the classifier phase. Both datasets were big and so, the task of analyzing the data in a thorough and manual way is a tedious and expensive one. Because of this, we developed a generic methodology to analyze the packets in a more efficient way, given the size of the datasets. This is done by using tools such as editcap and tshark, that are capable of dividing and filtering pcap files.

When analysed in Wireshark, from the testing dataset packets we could see come HTTP as well as some TCP packets being exchanged. And we can see that it loaded around 5 million packets. Analyzing this by hand would be hard and time-consuming. The training dataset on the other hand, was impossible to open on our machine as it occupied too much memory.

From this quick analysis of a trace, we gathered some information that could be useful in using on the classifier such as host names, http connection information and even patterns on the urls that the computers requested. This information allied with a couple of flow-based features tested in the literature (and new ones that we suggest) gave us a good start and allowed us to begin constructing the rest of the system.

3.4. Feature Extraction

For the machine learning part of our framework, we only considered flow features since they can capture botnet behavior without having to extract specific information of each machine. For example, we could train the classification algorithm with host characteristics like the IP address of the hosts or the ports with which they are establishing a link. What could potentially happen is that when we apply this system to other types of bonets or different IPs, these kinds of features would turn out to be useless since it captured similarities only on that specific network with those specific IP addresses. With that in mind, we will only consider features that allow us to classify not only previously seen and trained botnets but also new and unknown threats.

The features that were considered and evaluated in the framework can be found in Table 1. The features refer to flow characteristics of the traffic such as flow size and flow duration that identify a specific connection and not information that is relative to one machine or another.

Features
Flow Size
Flow Duration
First Packet Size
Average Packet Size
Number of Packets Exchanged
Percentage of Outgoing Packets
Number of Small Packets
Percentage of Small Packets
Average Payload Size
Host Entropy
HTTP GET URL Entropy
Biggest Packet Size
Average Inter Arrival Time

Table 1: Features tested

Most features presented here were already tested in the literature in detecting centralized and decentralized botnets. More specifically, from the article of the dataset [2], the authors use some of these features and discuss their usage since they have brought good results in the past. On the other hand, we introduce here new features that could help in training the classifier.

- Host Entropy
- Entropy of the HTTP GET URL
- Biggest Packet Size

The first two features are based on characteristics of HTTP connections with the idea that the host names that the machines exchange packets with can sometimes be easily identifiable. This is because the host name has the characteristic of being composed of random alphanumeric characters. From this idea, we can employ an Entropy function to the string and extract information from there. In information theory, entropy was first introduced by Shannon in [9] and can be defined as the expected number of bits of information in a message or string. It can be expressed by the following formula:

$$H(X) = -\sum_{i=0}^{N-1} p_i \cdot log_2\ p_i \qquad (1)$$

Where p_i is the probability of each character in the string. If the string has characters that appear

[4] https://www.wireshark.org
[5] https://scapy.net/

frequently, the entropy value will be much higher than a string with a more variety and uniqueness of characters. This is good for cases where the hostname does not repeat characters.

The second feature tested, rests on this idea too, but it is a lot more difficult to make it work well in practice. The entropy of the GET url could work because bots tend to mask information on the url such as using base64 encoding on the requests. The problem is, normal websites with usual requests might have high entropy as well when the url arguments are cookies for example. Nonetheless, we test this feature too so that we can discuss it later in Chapter 4.

Finally, we added a last feature that is based on the biggest packet found on the connection. Sometimes, to update the botnet for example, the infected machine will download an executable file to install on the host with malware. Since these executable files can be big, this feature is useful for that type of cases.

The extraction of features is done with a helper Python script called features.py which basically receives as input the list of flows (.flow file), splits each flow into feature values and chooses the features to be evaluated constructing a list. In addition to the feature vector, we have to construct a vector that will store all the correct prediction for each flow. This is possible because we have knowledge about the specific IP addresses that are malicious in the network. For the labels, we will be using a binary classification that is, if the vector we are labeling is considered malicious, the label associated is 1. Else, the label will be 0.

3.5. Classification

As stated in the sections above, we used the scikit-learn library on the implementation of our system since it simplifies our work by providing us with all necessary machine learning algorithms. Initially, to test out our code, we started by doing the pre-processing phase with only one kind of botnet. From there, we expanded the implementation to process all botnet data that we can find on the datasets.

We tested our framework with several popular classifiers as such as Support Vector Classifiers, Naive Bayes, Logistic Regression, Decision Trees, and Neural Networks.

For this phase of the system, we created a Python script called *classifier.py* which is the core of the framework. It takes the feature vectors created in the pre-processing phase and trains all the different classifiers that we want to test. After that, it tests the classifiers with new data and outputs the results to be evaluated. Also, the created classifier script implements the class that stores the feature data as well as a class that implements methods that wrap the scikit-learn package.

3.6. Evaluation

On this step of our detector, all the methods to output the results are implemented in the *classifier* script file and it wraps some scikit-learn methods that allow us to be able to calculate the metrics that we want to analyze. The metrics used will be discussed in the next chapter where we will support our results with graphics. In a first approach, we decided to test multiple classifier algorithms by analyzing their results on some main metrics such as precision, recall, f1-score, confusion matrix and accuracy. Then, we focused our experiments on the best classifiers. In our experience, Random Forests, Decision Trees, Linear Discriminant Analysis, and Neural Networks were the best algorithms tested. Forests and Trees performed well and had the fastest training and testing time, making them suitable for the detection of abnormal activities in the least time possible.

To test the classifiers, we used cross validation with a variable value for the folds to be trained. We also tested the classifiers without cross validation where we let an unseen set of feature vectors to be classified that belonged to the testing dataset.

3.7. Decision

The implementation for this last part of the system is actually quite simple. The information provided by the Evaluation component basically will serve to calculate the ratio of correctly predicted connections on each machine. This gives us an idea about what were the machines that were more flagged as malicious by the classifier. On the other hand, this ratio will give us the option to define a threshold on the Decider so that we can eliminate certain IP addresses that may not seem dangerous for the classifier. Also, machines with few flows might not be considered here because of the lack of information.

In the end, based on the above data, the system will output the IP addresses that seem like the most likely to be bots or engage in abnormal activities. This component is an example of a practical application of this type of detection system because, in a real world scenario, we want to be able to know how to apply the classification and the algorithms here described. The tool created with this work was made to be extensible. In this particular module, a lot of work can be done, as it is very simple as is. In the last Chapter of this document we will discuss ways on how to improve this, as well as give suggestions to future implementation options.

4. Results

As part of the development of the project, we tested the system in iterations as we implemented the solution. We gradually executed more tests as more features would be created in order to understand if the solution could be viable in the real world and if it could be improved compared to other work already done on the field.

The tests were conducted in a machine with Ubuntu 18.04, 8GB of RAM, and an Intel Core i5-7200 processor. The processing of files was executed in a Virtual Machine with 100GB of RAM because, as we will see in the next section, the data needs too much space to be stored on the original machine. We mainly tested the machine learning algorithms employed but we also tested the capacity of the system to perform as fast as possible and with fewer resources spent. The Evaluator component from the architecture, as seen in the previous Chapter, carries out a lot of the calculations for the tests using the methods from the scikit-learn and scikit-plot[6] package. Other tests, such as resources needed and time elapsed, are executed simply with code written in Python and Bash.

4.1. Data Processing

One of the most important aspects of the framework is the way that we treat the input data of our system. If we think about it, in reality, analysts deal with a tremendous amount of logs that come out of multiple defense mechanisms in a not normalized way. We present here the results of our data processing in terms of speed and memory needed to transform the pcap files into objects that we could manipulate to our needs. Table 2 shows for each dataset file used, the time needed to load it and the memory occupied in the system. Notice that the training dataset was split in two. This is because, on our system, we did not have enough memory to store all the structures of the training data. Because of this, we had to split the file in half (approximately) using the editcap tool in order to process all data.

File name	Size	Time elapsed	Memory occupied
training1.pcap	2.6 GB	1h 32m 52s	≈ 67.6 GB
training2.pcap	2.5 GB	56m 48s	≈ 41.1 GB
original_testing.pcap	2.02 GB	1h 16m 48s	≈ 55.1 GB

Table 2: Time and memory needed for each dataset

As we can see from the table, both the time and memory used is simply too high. The time metric is the most important here because, in order to make a viable analysis framework, we should be able to load the data and train it as fast as possible so that any threat could be stopped as soon as the framework finishes the classification.

To improve this results, one could use multithreading. With multi-threading, we could break the data into multiple files and parse the packet information with multiple processor threads. This way, we could speed up the processing phase. And since the flows don't depend on each other when parsing, we do not need to worry about the order of the threads finishing. In terms of memory consumed, there is not much we could do here since the Scapy package creates a lot of structures to parse the sessions into memory. This occupies a considerable size on memory so, the machines that do this task should have a lot of resources to deal with this problem. In some cases, splitting the dataset files here could be a solution too, but the data have to be processed in a sequential way.

4.2. Classifier Comparison

For Classifier selection, we conducted a couple of experiments to find out the best algorithm that suited the needs of the system, before even trying to make a decision about what machines should be further investigated. To do this, we tested the algorithms implemented in the library such as Neural Networks and Decision Trees regarding the metrics already discussed such as Accuracy, Recall, and Precision. We first wanted to know, if the features employed were valid and presented good results regarding the multiple botnets contained in the logs. To this end, a couple of botnets were tested separately first. We consider this experiment to be the most relevant and will discuss more around it because it is the one that is closest to what happens in real world setting. In a normal attack, we won't see a dozen of different botnets attacking. It is much more normal to have in a network, a big quantity of normal flows and then, only a couple of malicious flows, one usually from a single botnet.

On a second phase, we tested the whole dataset to see how the system would perform. Also, the majority of threats found on the testing dataset are not present in the training data. This will allow us to know if the system performs well when faced with unknown threats.

4.2.1 First Experiment - Analyzing a botnet at a time

In a first phase, as stated above, we tested the system in a more simple way by filtering the dataset flows to only a specific botnet at a time. More precisely, in the training dataset, we will use normal connections and three different botnets (Neris, Virut and RBot). In the test dataset, we will only leave flows of one specific botnet plus normal connections.

We present here the result for one botnet present on the training dataset as well as tests for one unknown botnets that do not appear in the training phase of the classifier. In terms of flow sizes in this set of tests, we tested the classifiers with exactly

[6]https://github.com/reiinakano/scikit-plot

60006 normal connections.

In Table 3 we have the results for a botnet that can be found in the training dataset, Virut. This specific test was evaluated with 33102 Virut connections.

Classifier Algorithm	Accuracy	Precision	Recall	F1-Score	Training	Total
Linear Discriminant Analysis	92%	94%	84%	88%	0.2s	0.47s
Linear SVC	90%	88%	82%	85%	31.15s	31.42s
Naive Bayes	49%	41%	91%	56%	0.09s	0.3s
Gaussian Naive Bayes	44%	38%	92%	53%	0.1s	0.34s
Logistic Regression	91%	90%	86%	88%	3.55s	3.81s
Neural Networks	92%	86%	91%	88%	4.03s	4.41s
Decision Tree	94%	87%	96%	91%	0.71s	0.93s
Random Forests	97%	93%	99%	96%	1.3s	1.54s

Table 3: Results for the different algorithms in detecting the Virut botnet

As we can deduce from the above table, the classifier performed quite well regarding the chosen features. Almost all classifiers performed well with the exception of the algorithms based on Naive Bayes. In terms of training flows, Virut had only 863 connections. The great results from Virut might have to do with the size of the connections found, making it probably easier to detect.

Regarding unknown botnets, we show here the test for Menti. Table 4 show the results of the classifier in detecting these this botnet.

The results above are slightly different from the first table. The values are slightly lower than from the first botnet showed, but the Recall values and Accuracy are still higher for some algorithms, meaning that the detection is working in detecting malicious unknown flows.

Overall, the classification system did a good job of detecting a single botnet at a time. As expected, the detector performed better when the experiment involved a botnet present in the training dataset although the results were not bad in the second experiment, reaching high values in the Accuracy and Recall metric.

Finally, we show here one more statistical metric associated with the classification that allow us to analyze the results even further. The ROC curve that plots the True Positive rate in the Y axis and the False Positive Rate on the X axis. The ideal in this case is to maximize the True Positive rate while minimizing the False Positive rate.

In both Figures 2 and 3 we can see examples of optimal ROC curves, where the area below the lines

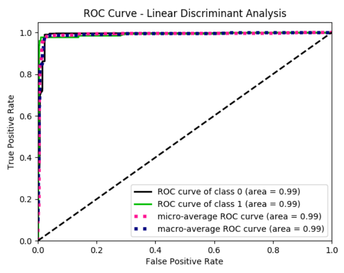

Figure 2: ROC curve of Menti with LDA

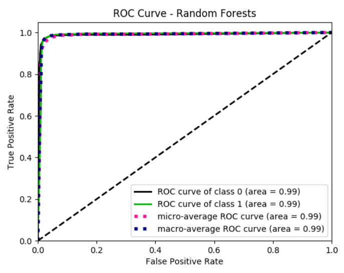

Figure 3: ROC curve of Virut with RF

are almost in the maximum value. These are very good results for these two botnets, meaning that we have a very low False Positive Rate for both of them.

4.2.2 Second Experiment - Analyzing the entire dataset

On the second experiment, we tested the whole dataset to see how the system would behave when facing multiple threats at the same time. While this approach might be less real and practical in the real world, we thought it was worthwhile to present the results nonetheless. Table 5 shows the results in testing the whole dataset.

From the table, we can make some conclusions. The first one is that the system does not perform very well when encountered with multiple botnets that operate very differently. For example, some botnets communicate via IRC and other through HTTP or UDP. It is extremely difficult to make a universal set of features that captures all these behaviors at once. The second conclusion is that

Classifier Algorithm	Accuracy	Precision	Recall	F1-Score	Training	Total
Linear Discriminant Analysis	97%	71%	98%	82%	0.21s	0.41s
Linear SVC	84%	30%	94%	45%	31.7s	31.9s
Naive Bayes	31%	9%	95%	16%	0.09s	0.26s
Gaussian Naive Bayes	23%	8%	99%	15%	0.11s	0.29s
Logistic Regression	95%	58%	98%	73%	3.72s	3.92s
Neural Networks	94%	55%	97%	70%	5.74s	6.02s
Decision Tree	92%	47%	98%	63%	0.73s	0.89s
Random Forests	97%	68%	98%	80%	1.21s	1.41s

Table 4: Results for the different algorithms in detecting the Menti botnet

Classifier Algorithm	Accuracy	Precision	Recall	F1-Score	Training	Total
Linear Discriminant Analysis	51%	62%	68%	65%	0.3s	0.75s
Linear SVC	65%	67%	97%	79%	58.8s	59.3s
Naive Bayes	71%	72%	93%	81%	0.16s	0.66s
Gaussian Naive Bayes	68%	68%	99%	80%	0.18s	0.71s
Logistic Regression	51%	62%	68%	65%	3.2s	3.72s
Neural Networks	60%	84%	51%	63%	4.9s	5.6s
Decision Tree	56%	79%	47%	59%	1.1s	1.61s
Random Forests	57%	82%	46%	58%	2.5s	3.2s

Table 5: Results for the different algorithms when testing all dataset

the large number of different patterns and communications in relation to the selected features, cause the algorithms to overfit. This is evident in cases where the algorithms tend to classify one label and not the other. For example, the Linear SVC detected almost all botnet connections. The problem is that it did not detect normal connections at all, with only 114 detected out of 60006.

4.3. Cross Validation

On this subsection, we will test the same classifiers with the same features and using the two already employed approaches: singular botnet at a time, and whole dataset but using the cross validation strategy. We will show the scores of each classifier in regards to the accuracy of the algorithms. We will do this by comparing the learning curves between each classifier. The learning curves have two purposes: to run the cross validation test on each algorithm and to show, by incrementing training samples, how the algorithm behaves with different quantities of information.

The learning curve shows us a green line corresponding to the mean accuracy score of the test samples while the red line corresponds to the mean accuracy score of the training samples. The green area and red area correspond to, respectively, the standard deviation of the testing and training scores computed in the various iterations of the cross validation.

With the exception of Linear SVC, the cross validation strategy was made with 10 splits, meaning that, the training dataset is split into 10 different training samples and 10 different testing samples. For each iteration, the classifier is trained with one of the training samples and tested with one of the testing samples. In the end, the accuracy score of the classification is averaged and this value is plotted on the figure. The cross validation is then repeated 5 times, with more training examples as seen in the X axis of the graphics.

In terms of the method to split the dataset, we used the strategy of time splits using the TimeSeriesSplit that scikit-learn provides. This strategy allows us to split the dataset into time intervals without shuffling the array of features. We want to do this because the training should not be done with future data. That is, the flows and the connections that appear on the network traces are not independent and using the flows that appear in the future might influence the results and create bias in the detector.

The learning curves are good tools in assessing if the testing scores of the algorithm are stabilized or not. They allow us to see if the number of samples in the dataset bring enough relevant information to achieve optimal accuracy scores for that classifier.

4.3.1 Analyzing a botnet at a time

From the experiment of analyzing a single botnet at a time, we will choose here only one botnet to show the results of the cross validation test since for each botnet, we have to show the graphics for all a lot of classifiers. We decided to showcase the Virut botnet, because it had great results before with so little flows trained.

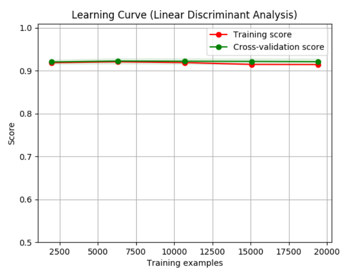

Figure 4: Learning curve for LDA - Virut

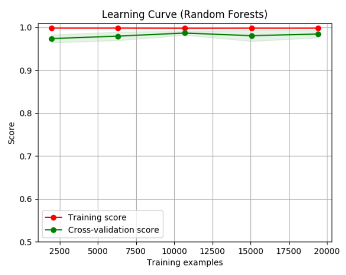

Figure 5: Learning curve for RF - Virut

From the figures generated by the cross validation experiment, we can conclude that the results are very good and encouraging. The algorithms showed performed very well and we can see that by the closeness of both lines in the figures. When both the test score and training score of each iteration of the cross validation (each point in the graph) are almost overlapping, we could say that we reached the optimal result for the classifier. For example, in the case of the Random Forests, the two lines are very close, where the Accuracy scores higher than 90%. These results only reinforce our conclusion that the system is very good in more practical scenarios, since it displays outstanding results in the detection of singular botnets.

4.3.2 Analyzing the entire dataset

The following graphics show the learning curves for two algorithms where the dataset is composed of all testing botnets found in the network traces, mixed in with normal activity connections. This experiment is similar to what can be found in the work of the Belgi *et al.* [2] who created the dataset used in our system.

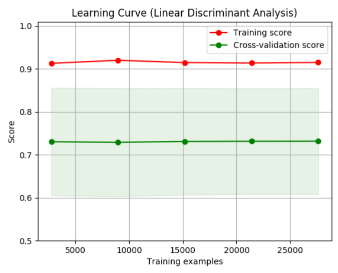

Figure 6: Learning curve for LDA

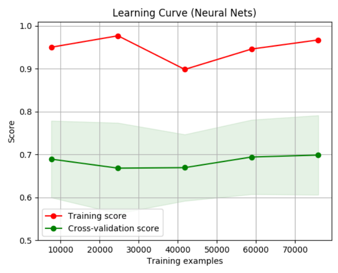

Figure 7: Learning curve for Neural Networks

From the figures generated by the cross validation experiment, we can conclude that the results are more or less in line with the classification of the whole dataset as shown in the previous subsection. And, unlike the previous experiment of a single botnet, the results are worse. However they seem to have improved slightly from the values seen on Table 5. In some cases, we can see that it could be possible to improve the results of the classification when adding more training examples. This is the case for all algorithms except Gaussian Naive Bayes and Naive Bayes, like we have seen on the previous cross validation experiment. The figures reflect that possibility when the training score is much higher than the testing score and it is not declining, meaning that, if we could execute more iterations of the cross validation with more training connections, the results would stabilize and improve a bit. However, the results could improve or decline, and this is deduced by the green area present on the figures. As stated before, the green area represents the range of values for the scores of the cross validation. So, in theory, the optimal scores could be in that area. To be completely sure and make conclusions, we needed to have more data to compute more iterations of the cross validation test.

4.4. Decider Results

The last component of the system, the Decider, aims at making a sense of the results from the evaluation of the classifiers in order to pick the most suited candidates of being infected and executing malicious activities such as data exfiltration. Here we present the results for both scenarios, as the rest of the evaluation process.

In the case of a single botnet scenario, we showcase Menti here. For this experiment, we considered a value of 0.8 for the *threshold* and 300 for the *flowThreshold* variables of our algorithm. Table 6 contains the results of running the Decider on a couple of chosen classifiers such as Neural Networks, Decision Tree, Random Forests and LDA.

Classifier	IP address	Correct Predictions	Wrong Predictions	True Label
LDA	147.32.84.150	4434	105	Botnet
	66.161.11.90	6	671	Not Botnet
Neural Netwoks	203.82.202.164	39	734	Not Botnet
Decision Tree	147.32.84.150	4335	204	Botnet
	195.228.245.1	49	596	Not Botnet
Random Forests	147.32.84.150	4539	180	Botnet

Table 6: IP addresses flagged by the Detector in the first scenario

As we can see from the 4 iterations (4 classifiers) of the decider, we notice that the Menti botnet is always correctly flagged given the thresholds we defined. From this result, we have got a running counter in an auxiliary dictionary that counts the times an IP was flagged during the algorithm. If we count the number of times the same machine appears, the conclusion is that the IP 147.32.84.150 has the strongest probability of being infected since it appears four times in the above table. This result is correct because that IP address belongs to the Menti botnet.

The second scenario is much more complex and hard to analyze. Since the datasets have more than 500 different IP addresses, and given the results of the second scenario in other tests, it is clear that we are going to wrongly flag a lot of machines. With the same parameters as above, there are only 4 different malicious machines that are flagged while only 2 have a high running counter. But so does a lot other non malicious machines. If we lower the *threshold* to 0.5, we flag a lot more bots but, at the same time, we catch more benign machines.

What we can conclude from this is that, the Decider component performs best in the scenario of a singular botnet in the traffic because the results of the classifier were very good there too. When the whole dataset is considered, we flag too many IP addresses to make this application viable.

5. Conclusions

In this work, we defined the data exfiltration problem and discussed some techniques that allow attackers to steal data as well as defense mechanisms that exist today. We discussed why some of the current defense mechanisms are not enough because there are multiple ways to exfiltrate data. Also, organizations sometimes overlook the covert aspects of data exfiltration and use tools that can be outdated, not dealing directly with this type of problem.

We described a simple taxonomy for data exfiltration techniques found in [3] and created a survey. The survey was divided into four subsections: Exfiltration Techniques, Exfiltration Detection Mechanisms, Botnet Data Exfiltration, and Machine Learning. In regards to the exfiltration and based on the taxonomy, we explored four main categories: Network, Physical, Covert Channels, and Steganography. This division is quite flexible because some previous work mix network techniques with covert channels but it gives us a good starting point on the topic.

Then, we proposed a solution to the problem of detecting botnet activity using a machine learning approach. The goal is to observe what are the characteristics and behaviors of botnets and extract features allowing us to train a classifier. We showed that our system was composed of five main modules: Data Collector, Feature Extractor, Classifier, Evaluator and the Decider. The first two components allowed us to conduct the pre-processing phase on the pcap log files, transforming them into arrays that we could manipulate and parse features to use on the classification. In the next step, we used multiple classifiers and evaluated them in order to get the best results. The Evaluator module produced the report from the output of the classifiers that we discussed in Chapter 5 and gave us a good picture of the viability of the system in detecting single and multiple families of botnets. Finally, we saw that the Decider component analyzed the results from the Evaluator and gathered information from the Feature Extractor to choose the most likely infected machines.

In the end, we executed some experiments and discussed scenarios that the system could be faced with. In regards to the data processing, we discuss the long time needed to process the log files as well as the space occupied by the structures created by Scapy that could make the system less practical. On the other hand, the results from the classifier were encouraging when the system was faced with singular botnets (accuracy around 97%), specifically, those that we already had knowledge off. When we tested all the botnets found in the dataset, the detection rate was lower when using the cross validation strategy, reaching 73% accuracy. Finally, we checked the results on the Decider component. The results were best when we executed the algorithm with a single botnet by marking the running botnet correctly, In terms of the whole dataset, the Decider had very trouble in flagging correctly the machines, although that task is harder and arguably unrealistic.

5.1. Future Work

The work on detecting data exfiltration and botnet activity, in general, is far from over, and there is still a lot to be researched and experimented in order to make these kinds of machine learning systems more efficient and effective in preventing attacks as fast as possible. Overall the results of our project were very good in detecting instances of singular botnets at a time. In the case of known botnets, the cross validation experiment showed us the potential for this system where we saw that the classifiers reached optimal Accuracy and Recall scores in some cases, all above 90%.

When testing with singular unknown botnets, we reached good accuracy results but when faced with very few malicious connections to classify, there is still a lot of false positives to go through and analyze. The classifier module of the system could be more fine tuned by testing different-features for example or, by employing a more sophisticated algorithm to make classifications, like a consensus style procedure that selects the best algorithm for each specific family of botnet.

Our system is slow on the data processing side although it could perform a lot better with the methods we discussed. In the future, the time spent in this task should be lowered to speed up large processing of network traces.

The problem of detecting data exfiltration by botnets was not directly solved by our solution in particular and it could be addressed in the future. The fact that a dataset meeting the specific criteria that we wanted did not exist, made the practical application harder to execute. In further research, it would be beneficial for the area to develop logs and data to create a dataset in data exfiltration or, to use already existing file access logs and feed them into our system as an extension to our tool.

Finally, the Decider module is a good bridge between the classifier results and a real world application. This component could be further improved by creating a better score to detect malicious machines in a network. Suggestions here could range from gathering information from multiple data sources to compute a score or to use some kind of heuristics on the features explored.

References

[1] A. Al-Bataineh and G. White. Analysis and detection of malicious data exfiltration in web traffic. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, pages 26–31. IEEE, 2012.

[2] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani. Towards effective feature se-

lection in machine learning-based botnet detection approaches. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 247–255. IEEE, 2014.

[3] A. Giani, V. H. Berk, and G. V. Cybenko. Data exfiltration and covert channels. In *Defense and Security Symposium*, pages 620103–620103. International Society for Optics and Photonics, 2006.

[4] J. Grier. Detecting data theft using stochastic forensics. *Digital investigation*, 8:S71–S77, 2011.

[5] G. He, T. Zhang, Y. Ma, and B. Xu. A novel method to detect encrypted data exfiltration. In *Advanced Cloud and Big Data (CBD), 2014 Second International Conference on*, pages 240–246. IEEE, 2014.

[6] Y. Liu, C. Corbett, K. Chiang, R. Archibald, B. Mukherjee, and D. Ghosal. Sidd: A framework for detecting sensitive data exfiltration by an insider attack. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10. IEEE, 2009.

[7] A. Nadler, A. Aminov, and A. Shabtai. Detection of malicious and low throughput data exfiltration over the DNS protocol. *CoRR*, abs/1709.08395, 2017.

[8] P. C. Patel and U. Singh. Detection of data theft using fuzzy inference system. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 702–707. IEEE, 2013.

[9] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[10] S. Venkatesan, M. Albanese, G. Cybenko, and S. Jajodia. A moving target defense approach to disrupting stealthy botnets. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 37–46. ACM, 2016.

[11] S. Venkatesan, M. Albanese, A. Shah, R. Ganesan, and S. Jajodia. Detecting stealthy botnets in a resource-constrained environment using reinforcement learning. In *Proceedings of the 4th ACM Workshop on Moving Target Defense*, pages 75–85, 2017.

[12] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, 2013.