A Comparison of Machine Learning Approaches to Detect Botnet Traffic

Brendan Abraham*, Abhijith Mandya†, Rohan Bapat†, Fatma Alali‡, Don E. Brown*, Malathi Veeraraghavan‡
*School of Systems and Information Engineering, {bea3ch, deb}@virginia.edu
†Data Science Institute, {am6ku, rb2te}@virginia.edu
‡Department of Electrical and Computer Engineering, {fha6np, mv5g}@virginia.edu
University of Virginia, Charlottesville, Virginia 22903

Abstract—**During the past decade, botnets have become one of the most significant threats in the field of network security. A botnet attack typically works by infecting a device with malware and then recruiting it into a network of infected devices controlled by an attacker, which may lead to severe economic and social consequences. As a result, a considerable amount of research has been conducted to detect and prevent such attacks.**

In this paper, we create a foundation for an anomaly-based intrusion detection system to improve network security and to help reduce human involvement and error. The network traffic we used is captured in the form of connection logs, which are generated by a popular network monitoring framework called Bro. We use our proposed framework to compare the performances of multiple supervised learning approaches, including Logistic Regression (LR), Naive Bayes (NB), Support Vector Machine (SVM), Random Forest (RF) and Neural Networks (NN) at anomaly detection. We evaluated these models using F1 Score and Area Under Curve (AUC). Our models are trained on malicious network traffic samples from pervasive botnets like Zeus, Miuref and Conficker, as well as benign traffic samples. Using traditional cross validation, we illustrate that Random Forest has the best performance for anomaly detection.

To test each algorithm's ability to generalize to unseen bot types, we implemented a custom-designed Leave-One-Out Cross Validation (LOBO-CV). In this procedure, each algorithm is trained on all but one bot family and evaluated on their ability to detect the unknown botnet traffic. We then improved overall detection performance by ensembling our two best models. Our results demonstrate we can detect both previously seen bot families and unseen botnet families with a high degree of confidence.

I. Introduction

Cyber-attacks cause billions of dollars in losses each year and this cost is projected to grow to USD 2.1 trillion by 2019 [1]. Despite the speed, variability, and growing commercial implications of breaches and cyber-attacks, institutions have been slow in implementing robust countermeasures against them. These cyber-security vulnerabilities were exploited recently by a ransomware, WannaCry, which caused global financial and economic losses estimated to the tune of USD $4 billion [2].

Cyber-intrusion is one of the forms in which cyber-crime manifests. The growth of cyber-crime has also resulted in cyber-intrusions becoming more commonplace, more dangerous, and more sophisticated. To prevent cyber-intrusions, institutions have adopted Intrusion Detection Systems (IDS). An IDS is a device or software application that monitors a network for malicious activity or policy violations.

There are two types of IDSs: signature-based and anomaly-based. A signature-based IDS operates in a similar way to a virus scanner by searching for known identities (or signatures) of each specific intrusion event. While signature-based IDSs are efficient at identifying known attack patterns, they depend on receiving regular signature updates to maintain awareness of the most novel attack techniques. In other words, a signature-based IDS is only as good as the recency and range of its database of stored signatures. Anomaly-based IDSs create a baseline of traffic statistics and detect attacks through analysis of the variations in the traffic patterns relative to the baseline. Unlike signature-based systems, anomaly-based detection systems are always changing and learning from new data. In principle, they can provide adaptive countermeasures to neutralize the rapidly changing cyber threat landscape.

In this paper, we create a foundation for an anomaly-based IDS that can detect botnet traffic. We implement and evaluate five different machine learning algorithms on a dataset consisting of normal traffic and malicious traffic from eight different botnet families. Of all candidate models, Random Forest performed the best,achieving an F1 Score of 0.99. Feature importance analysis revealed that features pertaining to flag counts and packet counts have significant predictive power. We also performed a unique cross validation designed to measure the performance of our models against unseen botnet families. The Random Forest, suffers slightly, yet achieves a F1 score which is better than all other tested models.

The paper is organized as follows. Section 3 provides necessary background, section 4 describes our methodology and dataset, and sections 5 & 6 describe the modeling techniques and results.

II. Background

An important source of cyber-attacks is malware, or malicious software that infects and compromises a host machine. Malware proliferates in different forms, one of them being botnets, which are groups of remotely controlled, compromised machines. These compromised machines, called *bots*, connect to a central server operated by a *botmaster* that gives them instructions to execute. The bots periodically communicate with the botmaster to receive new instructions. This back and forth between bots and botmaster is known as Command and Control Communication*C&C*. A botnet can be leveraged to

978-1-5090-6014-6/18/$31.00 ©2018 IEEE

perform a variety of deadly tasks, such as DDoS (Distributed Denial of Service) attacks, APT (Advanced Persistent Threats) attacks, or phishing attacks [3]. Since they are a necessary tool for many different attacks, stopping them is a top priority from a security standpoint.

In every network connection, there is a source S (usually a client) and a destination D (often a server). The two communicate back and forth by sending digital chunks of information to each other. These chunks are known as *packets*, which are the smallest unit of data that can be sent across a network. Botnets, like normal traffic generate these packets during network communication. Each packet contains a header, which specifies S and Ds' IP addresses and a payload containing all the data to be sent to the destination. Each packet is associated with a certain *protocol*, or communication standard. In many protocols such as Transmission Control Protocol (TCP), hosts send *flags* to each other, which act as keywords that mark different stages of the conversation.

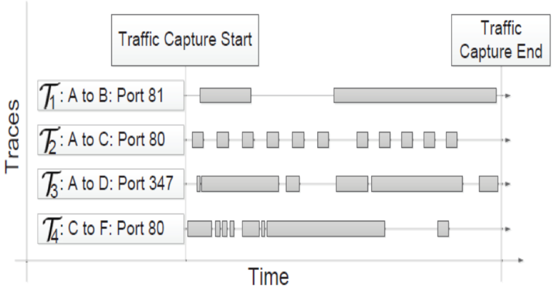

Fig. 1. An illustration of the features extracted from traces. Graphic courtesy Botfinder [4]

Packet traffic can be recorded in *pcap* (packet-capture) files and directly analyzed with open-source software. While pcap data provides highly granular information for rich analysis, it accumulates quickly and requires significant storage space. An alternative to pcap data is *flow* data, which contains aggregate level statistics about the packets exchanged between a source and destination over a certain time interval [4]. Since they are summary statics, they consume significantly less space than pcap data. However, they lose the detailed information contained in the payloads of packet-level data. Finally, flows can be further aggregated into *traces*, which represent all the traffic between S and D (See Fig. 1). Analyzing traffic at a trace level facilitates the computation of highly useful metrics for anomaly-based detection. In our work, we extract temporal and size features from traces between endpoints in order to detect malicious traffic.

III. Related Work

Different protocols can be used to communicate between the botmaster and the compromised machines. Ma et al. proposed a model to detect Internet Relay Chat (IRC) based botnets. Arshad et al. [5] and other researchers [6], [7] created models to detect both IRC and Hypertext Transfer Protocol (HTTP) botnets. In our models, we chose to focus only on HTTP-based botnets because the popularity of IRC botnets has degraded in recent years. Rossow et al. [6] analyzed ˜104,000 botnet samples from VirusShare and found only 8% of the samples used IRC as a communication channel while the other samples used HTTP and Domain Name System (DNS). Also, Zeus, one of the most pervasive botnets, uses HTTP. Zeus compromised 3.6 million PCs in the US alone [8]. HTTP botnets follow a pull approach, where the bot continuously polls the C&C server for updates and commands.

A considerable number of studies have been conducted to create models based on a limited number of botnet families, where a different model is created for each family, or one model is created for 2-3 botnet families. For example,Hadaddi et al. [9] developed C4.5 and Naive Bayes models to detect HTTP-based botnets using two families: Zeus and Citadel. Eslahi et al. [10] proposed the use of HAR (High Access Rate) and LAR (Low Access Rate) filters, which are designed to provide upper and lower bounds on suspicious periodicity. The model was based on two botnet families: BlackEnergy and Bobax. Wang et al. [11] created three clustering models for Kraken, BlackEnergy and Zeus. Lu et al. [12] combined network traffic analysis with hidden Markov models to differentiate the behavior of Zeus C&C communication from normal traffic. Torres et al. [13] used the same dataset we used but picked only two families (DonBot and Neris) to train the model. The authors of Botfinder [4] proposed a machine learning based system that calculates statistical features similar to the features we used. However, they trained a separate model for each type of malware family.

Some botnet detection models require inspecting packet payloads to extract the required information for detection [8], [14]–[17]. For example, Etemad et al. [14] proposed a method to separate IRC traffic from HTTP traffic by light payload inspection. Most of the feature extraction algorithms just regard malware detection as a regular machine learning problem and mainly aim at improving the final detection performance. However, these algorithms usually ignore an important aspect of malware detection, i.e. the robustness. If malware authors know how these algorithms work, they would adopt some pretense techniques accordingly in order to bypass these algorithms [18]. Our solution is based on trace level information that does not require payload inspection. Soniya et al. [19] trained a neural network classifier by running 120 botnet malware samples including Pushdo, Banbra, BlackEnergy, Sasfis, Bifrose, Dedler, and Zeus. Their model achieved a false positive rate of 2.5%. Bilge et al. [20] proposed a botnet detection system called DISCLOSURE that extracts three types of features from Netflow: size, host access pattern, and temporal behavior. The features are based on their hypothesis that C&C communication happens in predictable patterns that can be identified with behavioural and temporal metrics. However, while their final model achieved a false positive rate less than 1%, its accuracy was less than 65%. While our datasets are different, we demonstrate that it is possible to significantly improve the accuracy using similar features. Finally, Haddadi et al. [21] analyzed Netflow data from three Botnets Zeus, Conficker and Torpig, two of which are also cited in our work. Flow features such as duration, number of packets, number of bytes, flows and bits per second were used to classify malicious traffic. A C4.5 Decision tree model and a Symbiotic Bid-Based (SBB) model were trained and tested where the false positive rates were on average 5-10%, depending on the botnet family. We demonstrate that our best model can achieve a FPR of less than 1% on similar traffic.

IV. Methodology

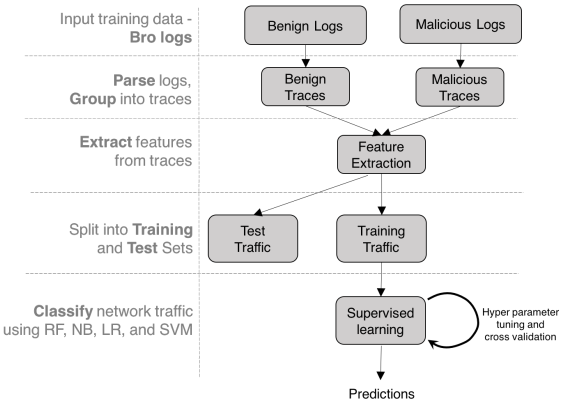
Input training data - **Bro logs**
Benign Logs
Malicious Logs
Parse logs, **Group** into traces
Benign Traces
Malicious Traces
Extract features from traces
Feature Extraction
Split into **Training** and **Test** Sets
Test Traffic
Training Traffic
Classify network traffic using RF, NB, LR, and SVM
Supervised learning
Hyper parameter tuning and cross validation
Predictions

Fig. 2. Pipeline for Botnet Traffic Detection.

Our approach to detect malicious traffic involved multiple stages, as shown in Figure 2. First, we collected malicious and benign traffic samples from Czech Technical University's (CTU) public repository of network traffic and converted each sample to a series of traces [22]. Then, we performed feature extraction on the traces and calculated statistics related to size, frequency, and duration of communication. Finally, we performed supervised learning on these new features. We performed traditional k-fold cross validation and hyper parameter tuning as well as a custom-designed Leave-One-Bot-Out cross validation to improve the performance of each model, including an emsembling that improved our outcomes. In the following subsections, we layout the stages in greater detail.

A. Data Description

Our data consisted of traffic from eight different bot families. We have used data released by the Malware Capture Facility Project under Stratosphere IPS Project [22]. The researchers used a testbed network topology consisting of a set of virtualized computers to create *malicious* and *normal* network traffic [23].

Traffic that came to or from any of the known infected IP addresses was labeled as malicious data. The researchers who generated this traffic used open source code specific to eight different families of botnets. In this way they produced malicious traffic from each of these families with correct, ground truth labels for each bot. Traffic from the known and controlled computers in the network, such as routers, proxies, or switches, was labelled as normal data.

The traffic was captured in different formats including pcap (packet capture), Netflow and Bro logs. For our analysis, we used conn logs, which contain flow-level data and are generated by the network monitoring framework Bro. Overall, our dataset contained over 22,000 traces and around 1.6 million flows.

TABLE I
Malware Description

Family	Malware Description	Year	Botnet Size
Zeus	First widespread banking trojan - used to steal banking information.	2007	3.6M
Conficker	Worm used to infect Windows systems - Has infected computers in over 190 countries	2008	200k
Dridex	Another banking trojan that targeted large American corporations from 2012-2015	2011	100k-500k
Necurs	Massive botnet used to launch ransomware attacks and banking trojans.	2012	6M
Miuref	Trojan that facilitates click fraud and downloads malicious content.	2014	100k
Bunitu	VPN scam that uses infected hosts as a proxy for remote clients.	2014	100k-500k
Upatre	Malware downloader that downloads and executes Dyre Banking trojan	2014	100k-500k
Trickbot	Trojan that leverages HTML and Javascript injections to steal banking information	2016	100k

TABLE II
Traffic Type Distribution in Dataset

Traffic description	Number of traces	% of overall
Normal traffic	13,514	48%
Bunitu	6,761	24%
Necurs	3,975	14%
Miuref	1,532	6%
Zeus	711	3%
Other malware	1,379	5%

Table I details the different botnet families part of our dataset, while Table II details the distribution of traces of these botnet families in our dataset.

B. Parsing Bro Logs and Trace Construction

As described above, we chose malicious samples from a variety of Botnet families to increase the robustness of our detection models. Additionally, as each sample was collected independently, so there were no inter-dependencies between samples. We exclusively used the connection logs (conn logs) for our pipeline, which contained flow-level data about each connection that occurred in the network. These records were grouped into traces containing all flows between the same four-tuple of source IP, destination IP, destination port and protocol.

C. Feature Extraction and Data merging

For every sample, each trace was collapsed down to a single record containing statistics characterizing the typical behavior between each source and destination on the network. The features we calculated can be generalized to three groups. The first set of features relates to the *volume* of the communication, or the average number of packets and amount of data sent in a typical flow. The second set of features pertains to the *timing* of communication, or the average flow duration and average time interval between successive flows. The final set of features pertains to the state history of connections in the form of flag counts. These counts were obtained by parsing the *state_history* field in the conn logs, which contains a string of successive flags sent throughout the connection. Capital flags were sent from the source, while lowercase flags were sent from the destination. We kept separate counts of source flags and destination flags with the hops of discovering some asymmetries in malicious communication. The features we used were as follows:

1) Average time interval between two consecutive flows in a trace
2) Average Number of source packets and destination packets
3) Average flow duration within a trace
4) Total flag counts (a/c/d/f/h/r/s/t) - Records the state history of connections as a string of letters. The definitions of these flags are as follows
    a) a - pure ACK, or Acknowledgement flag
    b) c - packet with invalid checksum
    c) d - packet with payload data
    d) f - packet with the FIN flag set, terminating the connection
    e) h - SYN+ACK (handshake) which is standard protocol for establishing a TCP connection
    f) r - packet with RST bit set which resets the connection
    g) s - SYN w/o the ACK bit set (or an open-ended handshake)

We used the mean and standard deviation of the first three metrics - Time interval between start times, number of source and destination packet and flow duration - as features for the training and testing datasets.

D. Modeling

We used the training dataset to create five binary classifiers that predict whether a given set of traffic was benign or malicious. We first attempted to create a baseline for classification performance by using a light-weight logistic regression model. We then added Naive Bayes, Support Vector Machine, Random Forest and Neural Network to our umbrella of supervised learning algorithms. In the next section, we explore our modelling process in detail and explain our model-of-choice. We further tested the generalization capability of these models across unseen botnet families which we have termed as Leave-One-Bot-Out cross validation (LOBO-CV). In this process, models are trained on all but 1 family, and the unseen bot is used as a test set. The same performance metrics are used as measurement. Further, using the best performing models, we devised ensemble to find a unified model to look for the best performance across all families.

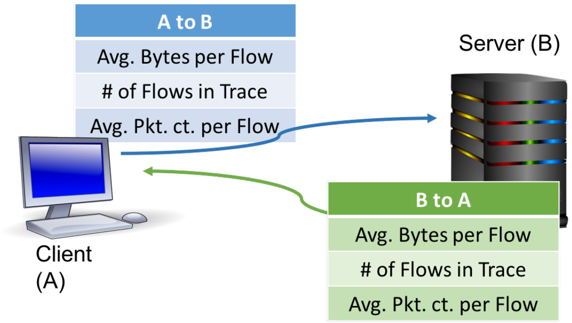

Fig. 3. An illustration of the features extracted from traces.

V. Modeling Techniques

We used five different supervised learning techniques to classify network data and measured their performance. These classifiers were chosen since they represent the most widely used supervised learning models today. Performance of each one varies widely and depends upon the complexity of the given dataset. Since this is a classification problem, prediction accuracy (percentage of correct predictions divided by the total number of predictions) was used for evaluation. This evaluation was carried out in two stages. First, the dataset was split into a training set which contained two-thirds of the traces, while the rest was used as the test set. The supervised learning models were trained on the training data and then cross-validated for model selection and model performance assessment. Next, the models were tested using the test dataset. The predictions on test data were used to arrive at the performance metrics for model comparison. All models were trained and cross validated using the Caret package in R [24].

A. Logistic Regression

Logistic regression (LR) is a widely used classification technique in which the probability of the outcome (trace maliciousness) is related to a series of potential predictor variables by an equation of the form

$$log[\frac{p}{1-p}] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_i X_i$$

where p is the probability of the maliciousness, β_0 is an intercept term, $\beta_1 .....\beta_i$ are coefficients associated with each variable $X_1 .....X_i$. This model assumes that all predictors are related in a linear manner to the log odds of the outcome, or in our case, maliciousness of the trace. To achieve these linear relationships, we performed log transformations on the predictors before modeling. This learning technique has been the method of choice till very recently and still affirms a baseline for statistical modelling. Although it is easy to implement, easy to interpret, and has low computational requirements, it is increasingly seen to be inferior to more recent and complex machine learning algorithms.

B. Naive Bayes

Naive Bayes (NB) is a widely used framework in statistical modeling using Bayes Theorem. Naive Bayes is the simplest form of a Bayesian network, in which all attributes are independent given the value of the class variable.

$$f_{nb}(E) = \frac{p(C=+)}{p(C=-)} \prod_{i=1}^{n} \frac{p(x_i C=+)}{p(x_i C=-)}$$

where C is the label of maliciousness and x are the predictor variables. This is called conditional independence. However, this assumption is rarely true in most real-world applications. Naive Bayes owes its good performance to the zero-one loss function. This function defines the error as the number of incorrect classifications. Unlike other loss functions, such as the squared error, the zero-one loss function does not penalize inaccurate probability estimation if the maximum probability is assigned to the correct class. This means that naive Bayes may change the posterior probabilities of each class, but the class with the maximum posterior probability is often unchanged. Thus, the classification is still correct, although the probability estimation is poor.

C. Support Vector Machines (Radial)

Support Vector Machines (SVM) separate a given set of binary labeled training data with a hyper-plane that is maximally distant from these binary classes (known as the maximal margin hyper-plane). For our case, where a non-linear separation is possible, they work in combination with a radial kernel, that automatically realizes a non-linear mapping to a feature space. The hyper-plane found by the SVM in feature space corresponds to a non-linear decision boundary in the input space.

$$\gamma = min_i y_i \Big\{ \big\langle w, \phi(x^i) \big\rangle - b \Big\}$$

where the margin γ is maximized, the hyperplane (w, b) with input data x_i with the corresponding label y_i. The quantity $(w, \phi(x^i)\big\rangle - b)$ corresponds to the distance between x_i and the decision boundary.

D. Random Forest

Random Forest (RF) is an ensemble of decision trees where the features of each tree are chosen by a random vector sampled independently and with the same distribution for all trees in the forest. The error rate for forests converges to a limit as the trees grow asymptotically. Every classification tree in the forest casts a vote for the sample after which the majority vote determines the class of the sample. The strength of the individual trees in the forest and the correlation between them determine error for random forest classifiers [25]. Internal estimates monitor error, strength, and correlation, which are used to measure variable importance. It also holds better sway in terms of identifying underlying variable interactions.

E. Fully-Connected Neural Network

In a fully-connected, feed-forward neural network (NN), the input data is sent through a series of fully-connected layers of neurons that ultimately produce a continuous output between zero and one. Each connection has an associated weight that represents the strength of the connection. The higher the weight, the stronger the connection. In the brain, a neuron ”fires” if the combined input received from surrounding neurons exceeds an activation threshold. Similarly, in a NN, a neuron in one layer will send a strong signal if the weighted sum of the input it receives from the previous layer exceeds a certain threshold. The strength of this signal is determined by an activation function which can take many forms. Original neural networks used Haveside step functions, but most networks nowadays use Sigmoid (logistic) or Rectified Linear Unit (RELU) which is a piecewise-linear activation function. In our neural network, we used sigmoid activation functions for all neurons. The architecture of the neural network we used is shown in Figure 4.

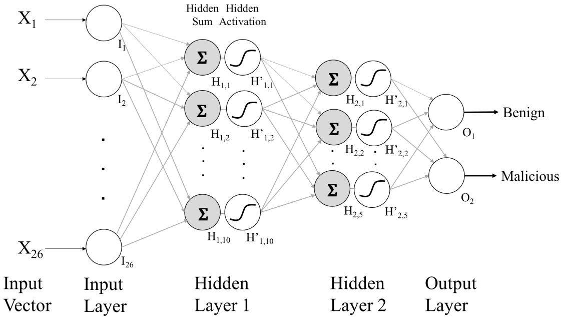

Fig. 4. An illustration of our neural network.

As shown in Figure 4, there are three types of layers: an input layer, two hidden layers, and an output layer. The two hidden layers have 10 and 5 neurons respectively. The input layer feeds the raw data to the nodes in the first hidden layer. Each node $H_{1,j}$ computes a weighted sum of its received inputs:

$$H_{1,j} = \sum_{i=1}^{n} w_{1i} x_i + b$$

where b is a bias term. The result is passed to a sigmoid function that converts it to a 0-1 scale:

$$H'_{1,j} = \frac{1}{1 + e^{-H_{1,j}}}$$

This process is repeated for the second hidden layer. Finally, a weighted sum of the second hidden layer's output is computed which assigns the malicious and benign probabilities. We trained the neural network with standard back-propagation and Stochastic Gradient Descent.

TABLE III
EVALUATION METRICS

Model	L.R.	N.B.	SVM	R.F.	N.N.
F1 Score	0.96	0.73	0.86	0.99	0.77
AUC	0.90	0.60	0.99	0.99	0.89

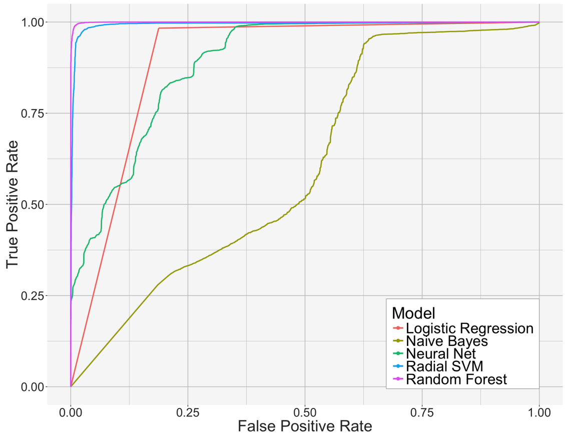

Fig. 5. ROC curves for candidate models.

VI. EVALUATION AND RESULTS

Our experiments clearly show that Random Forest is the superior model, achieving the highest F1 Score across all models of **X** (Table III). Moreover, the ROC curves show that Random Forest dominates all other models at every possible classification threshold. These results are competitive with those obtained by other detection systems such as Disclosure or Botfinder. SVM was comparable to Random Forest in terms of AUC but had an F1 score of 0.86. The next tier of models included Logistic Regression and the neural network. Logistic regression achieved a score of 0.96 and 0.77. We believe the neural net's performance would have improved if we had more data and trained it for more epochs. However, in both cases, the models missed a significant portion of malicious traffic, rendering them ineffective for real world anomaly detection.

A. Feature Importance

While Random Forest is the superior model in terms of performance, it remains to be seen which variables are driving this success. Figure 6 shows a variable importance plot generated by the Random Forest model. This plot measures the mean accuracy decrease when each feature is left out of the random forest model. The plot suggests that *mean_src_pkts* is the most important feature for Random Forest in discriminating between malicious and benign traffic. Additionally, certain metrics such as $mean\_intvl$ and $stdev\_intvl$ bear substantial importance in the Random Forest's decision making process.

Figure 7 contains boxplots of the top four features from the variable importance plot. There is clear separation in the $mean\_src\_pkts$ and $mean\_dest\_pkts$ boxplots, suggesting that on average, malicious traffic has a lower source packet count than normal traffic. Bots, typically will only contain basic information necessary for communication with it's control servers and vice-versa, thus on average, they'll contain lesser packets than normal traffic [4], [20].

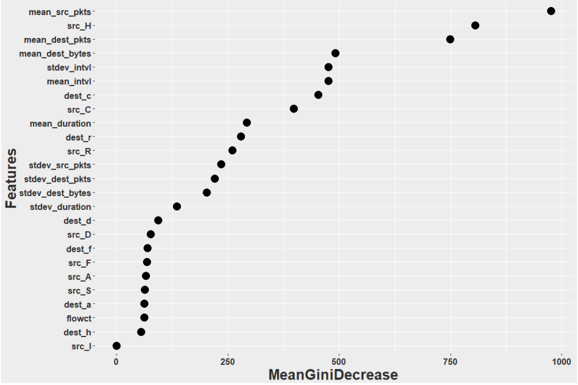

Fig. 6. Mean Accuracy Decrease when each variable is left out of the Random Forest Model

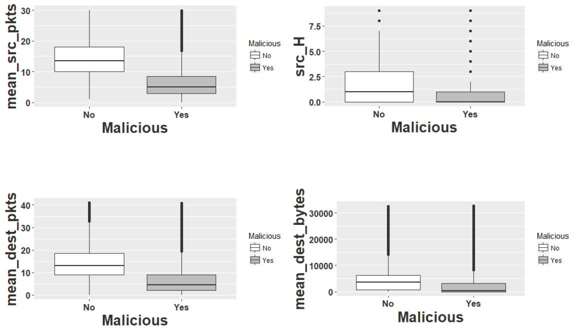

Fig. 7. Boxplots of the top 4 variables according to the importance plot

B. Leave One Bot Out Cross Validation

While the previous results are promising, they assume that botnet families are static over time. However, these families are ever-evolving, so it's likely that our system will encounter novel botnet families absent from the training data. Thus, it's imperative to understand how well our models classify an unseen bot. To measure this, we implemented a validation scheme called Leave-One-Bot-Out Cross-Validation (LOBO-CV). In contrast to traditional cross validation where the models have seen traffic from all botnets, each model in the LOBO-CV is trained on 7 bot families, and the 8th unseen bot is used as a test set. Unsurprisingly, all our models suffered performance-wise. Naive Bayes was the worst performing model while the Neural Network and Radial Kernel SVM performed comparably. SVM's score is notably lower than traditional cross-validation, suggesting it over-fit the data. Finally, Logistic Regression and Random Forest were the top performers. While the Random Forest scored greater than 0.99 on all but one bot family - Zeus. In contrast, Logistic Regression had its best F1 score of 0.95 on Zeus. This suggests that the algorithms were picking up on inherently different patterns in the data. This led us to believe that creating an ensemble of Random Forest and Logistic regression could improve overall performance.

TABLE IV
LOBO-CV F1 SCORE

Family	LR	NB	SVM	RF	NN	Family Average
Miuref	0.84	0.51	0.71	**0.99**	0.79	.82
Bunitu	**0.81**	0.47	0.72	**0.81**	0.71	.74
Upatre	0.94	0.57	0.90	**1.00**	0.95	.92
Dridex	0.84	0.53	0.85	**1.00**	0.81	.87
Necurs	0.82	0.48	0.91	**0.99**	0.82	.88
Trickbot	0.82	0.55	0.87	**1.00**	0.83	.88
Conficker	0.81	0.48	0.83	**1.00**	0.78	.85
Zeus	**0.95**	0.59	0.77	0.68	0.80	.74

C. Ensemble Learning

Based on the Leave-One-Bot-Out results, we decided to use ensemble modeling to improve the Random Forest's overall performance. To do this, we combined Logistic Regression and Random Forest predictions in a weighted fashion to produce ensemble probabilities. The predictions took the form

$$y_{ENS} = \alpha * y_{RF} + (1 - \alpha) y_{LR}$$

where α is a tunable weight parameter between 0 and 1, and y_{RF} and y_{LR} are random forest and logistic regression predictions respectively. These probabilities were converted to class labels (malicious or benign) via a cut-off threshold. Both alpha and the cut-off threshold were learned by using grid search to find the combination that yielded the highest F1-Score. Below is a comparison of the ensemble's leave-one-out performance to that of Random Forest and Logistic Regression.

TABLE V
LOBO-CV F1 SCORE FOR ENSEMBLE MODEL

Family	Ensemble	RF	LR
Miuref	0.99	0.99	0.84
Bunitu	0.90	0.81	0.81
Upatre	0.99	0.99	0.94
Dridex	1.00	1.00	0.84
Necurs	0.99	0.99 0.82	
Trickbot	0.99	0.99	0.82
Conficker	1.00	0.99	0.81
Zeus	0.95	0.67	0.95

Bunitu's F1 score rose from 0.81 to 0.89, while all other scores were the same. It seems the ensemble was able to use the best aspects of both models to increase or maintain detection performance for unseen botnet families. Since Bunitu formed the largest part of our data, we feel this ensemble has the potential to improve our predictive power when we encounter a larger, more variable dataset with unknown bot families.

VII. CONCLUSION AND FUTURE WORK

We achieved two main objectives through our work- first, we identified useful features to classify malicious traffic, and second, we compared the performance of five different supervised learning models. We found that $mean\_src\_pkts$ and the $meanandstdintervals$ play the most significant role in discerning malicious traffic.

Our results showed that Random Forest was the superior model with an F1 Score of bf0.99. Random Forest proved to be more robust than the other models, as the LOBO-CV results demonstrated it could generalize to most unseen bots. This is an important finding as normal network usage is highly sporadic and hence robust models are required for monitoring these networks. The Logistic Regression is a fantastic light-weight substitute. The results from the Neural Network were slightly underwhelming but we hypothesize that a larger training set, better topology and more number of epochs will improve it's performance. We also ensembled the Random Forest and Logistic Regression to produce overall performance that beat any singular model for all families.

While our results are encouraging, they might not generalize well on real network traffic as our datasets had a balanced distribution of malicious and benign traffic. In the wild, malicious traffic usually constitutes a small fraction of the entire dataset. This class imbalance will prove to be a challenge for traditional supervised learning models, requiring the creation of custom loss functions.

In future work, we plan to extend this approach to identify botnet traffic across a large network like the University of Virginia. There is evidence of temporal metrics like host access patterns and Fast Fourier Transforms (FFT) that can further improve detection [4]. We also plan on simulating bot attacks to produce rich data that can be used effectively train our models. Ultimately, we would like to setup an online, quick response system that can identify, trigger and quarantine botnet traffic.

REFERENCES

[1] "https://www.forbes.com/sites/stevemorgan/2016/01/17/cyber-crime-costs-projected-to-reach-2-trillion-by-2019/."

[2] "https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses/."

[3] S. Bano, "A study of botnets: Systemization of knowledge and correlation-based detection," 2012.

[4] G. V. C. K. Florian Tegeler, Xiaoming Fu, "Botfinder: Finding bots in network traffic without deep packet inspection," *CoNEXT '12 Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 349–360, 2012.

[5] S. Arshad, M. Abbaspour, M. Kharrazi, and H. Sanatkar, "An anomaly-based botnet detection approach for identifying stealthy botnets," in *Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on*, pp. 564–569, IEEE, 2011.

[6] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. Van Steen, F. C. Freiling, and N. Pohlmann, "Sandnet: Network traffic analysis of malicious software," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pp. 78–88, ACM, 2011.

[7] M. Graziano, C. Leita, and D. Balzarotti, “Towards network containment in malware analysis systems,” in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 339–348, ACM, 2012.

[8] T. Cai and F. Zou, “Detecting http botnet with clustering network traffic,” in *Wireless Communications, Networking and Mobile Computing (WiCOM), 2012 8th International Conference on*, pp. 1–7, IEEE, 2012.

[9] F. Haddadi, J. Morgan, E. Gomes Filho, and A. N. Zincir-Heywood, “Botnet behaviour analysis using ip flows: with http filters using classifiers,” in *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*, pp. 7–12, IEEE, 2014.

[10] M. Eslahi, H. Hashim, and N. Tahir, “An efficient false alarm reduction approach in http-based botnet detection,” in *Computers & Informatics (ISCI), 2013 IEEE Symposium on*, pp. 201–205, IEEE, 2013.

[11] B. Wang, Z. Li, D. Li, F. Liu, and H. Chen, “Modeling connections behavior for web-based bots detection,” in *e-Business and Information System Security (EBISS), 2010 2nd International Conference on*, pp. 1–4, IEEE, 2010.

[12] C. Lu and R. Brooks, “Botnet traffic detection using hidden markov models,” in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, p. 31, ACM, 2011.

[13] P. Torres, C. Catania, S. Garcia, and C. G. Garino, “An analysis of recurrent neural networks for botnet detection behavior,” in *Biennial Congress of Argentina (ARGENCON), 2016 IEEE*, pp. 1–6, IEEE, 2016.

[14] F. F. Etemad and P. Vahdani, “Real-time botnet command and control characterization at the host level,” in *Telecommunications (IST), 2012 Sixth International Symposium on*, pp. 1005–1009, IEEE, 2012.

[15] M. Eslahi, M. Rohmad, H. Nilsaz, M. V. Naseri, N. Tahir, and H. Hashim, “Periodicity classification of http traffic to detect http botnets,” in *Computer Applications & Industrial Electronics (ISCAIE), 2015 IEEE Symposium on*, pp. 119–123, IEEE, 2015.

[16] K. Li, C. Liu, and X. Cui, “Poster: A lightweight unknown http botnets detecting and characterizing system,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1454–1456, ACM, 2014.

[17] M. Grill and M. Rehák, “Malware detection using http user-agent discrepancy identification,” in *Information Forensics and Security (WIFS), 2014 IEEE International Workshop on*, pp. 221–226, IEEE, 2014.

[18] W. Hu and Y. Tan, “On the robustness of machine learning based malware detection algorithms,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1435–1441, May 2017.

[19] B. Soniya and M. Wilscy, “Using entropy of traffic features to identify bot infected hosts,” in *Intelligent Computational Systems (RAICS), 2013 IEEE Recent Advances in*, pp. 13–18, IEEE, 2013.

[20] W. R. E. K. C. K. Leyla Bilge, Davide Balzarotti, “Disclosure: Detecting botnet command and control servers through large-scale netflow analysis,” *ACSAC ’12 Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 129–138, 2012.

[21] A. N. Z.-H. M. I. H. Fariba Haddadi, Dylan Runkel, “On botnet behaviour analysis using gp and c4.5,” *GECCO Comp ’14 Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 1253–1260, 2014.

[22] “https://stratosphereips.org/category/dataset.html.”

[23] J. A. S.Garca, M.Grill, “An empirical comparison of botnet detection methods,” *Computers Security*, pp. 100–123, 2014.

[24] “https://cran.r-project.org/web/packages/caret/index.html.”

[25] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.