Traffic Classification for Flow Whitelisting

Marc Leef
Princeton University
88 College Road West
Princeton, NJ 08540
+1(503) 750-1809
mleef@princeton.edu

Ranjana Addanki
Princeton University
88 College Road West
Princeton, NJ 08540
+1(703) 774-7290
raddanki@princeton.edu

***Abstract*–Network Intrusion Prevention Systems (IPS) are widely employed by universities and other organizations to protect users from malicious attackers and potentially harmful traffic. Princeton University makes use of an IPS at the periphery of its network to filter all incoming and outgoing traffic from the rest of the Internet. The large volumes of information that must be processed and appropriately filtered by this device can cause a significant increase in latency, thus creating a performance bottleneck in the network. In this paper, we propose an algorithm to whitelist benign flows such that they bypass this system, reducing the stress on the device and improving network response times. We use data collected by the IPS over a one-week period to train a number of different classifiers (decision trees, random forests, and linear SVMs) to automate the execution of this whitelisting task. Both the accuracy and potential information savings of each of these classifiers is evaluated in detail. We conclude that decision tree classifiers provide the best combination of security guarantees and performance improvements, while highlighting the importance of low training and classification times for a successful deployment of this system in a real-world network environment in the future.**

1. Introduction

The Princeton University network consists of a wide variety of users, ranging from students and staff to faculty and researchers. With such a large and diverse set of users (and thus a large and diverse set of generated traffic), there are many possible malicious attack strategies and adversarial approaches that can go unnoticed. Thus, the Office of Information Technology (OIT) has set up a McAfee Intrusion Prevention System (IPS) at the border of the campus network and the Internet. OIT filters selected traffic as it passes across this border for a variety of purposes: protecting devices on the campus network from receiving traffic considered to dangerous to accept from the Internet, blocking communication protocols that have little legitimate non-local use, blocking traffic from specific IP addresses that have been sources of previous attacks, etc. However, forcing all outbound traffic to go through this device creates a system bottleneck that can significantly effect the throughput and latency of the network.

The IPS groups related packets into traffic units called flows. Flows that are determined to be benign are allowed to pass into/out of Princeton's Network, whereas flows determined to be malicious or adversarial are blocked and subsequently logged. The overhead associated with performing these classifications, and the single device, centralized nature of this system design, creates a situation in which innocent users can be met with potential slowness to maintain the integrity of the network.

The goal of this paper is to develop an algorithm that reduces the amount of traffic the IPS must process by preemptively identifying benign traffic. This subset of "good" traffic can be forwarded directly to its destination without stringent inspection. Specifically, we adopt a flow whitelisting strategy that aims to maintain the strong security guarantees provided by the IPS while reducing the degree of bottlenecking in the network. To achieve both of these goals, we use malicious flow data flagged by the IPS to create a reverse-engineered, but significantly lighter weight, system to separate the benign traffic from the malicious traffic.

We employ and evaluate the performance of three classifiers in this context: decision trees, random forests, and linear SVMs. Though it has been extensively documented previously [8] that applying machine learning techniques in the context of network traffic classification is an arduous task, we frame the problem in a subtly different way in an attempt to avoid some of these associated difficulties. Instead of trying to train our models to identify malicious traffic, we instead focus our efforts on elucidating the set of shared characteristics of benign traffic. In the context of our system, failing to correctly identify benign activity as such will simply result in closer inspection by the IPS and forego any potential data savings, while failing to correctly identify malicious activity could have much graver consequences. This subtle shift in learning methodology is not universally applicable, but certainly helpful in this specific situation.

The remainder of this paper is organized as follows. In Section 2 we describe related work in the areas of machine learning for traffic classification, analyzing network flow information, and strategies for malicious traffic identification. The algorithm design and traffic classification strategies are discussed in Section 3. Section 4 discusses the experimental results obtained by applying the various approaches described in Section 3.

2. Related Work

Utilizing machine learning effectively in the context of traffic analysis is an active area of research. While many previous works have made great strides in this area, overall improvement of model accuracy and creative feature engineering has been relatively stagnant. In this section, we highlight some of the most important recent works on this subject.

2.1 Machine Learning in Traffic Analysis

We plan on applying machine learning techniques to the traffic classification problem in our project. Previous works have demonstrated both the power and great difficulty associated with productively applying machine learning in this context. We hope to learn from and expand on these previous attempts over the course of our project to accurately and efficiently whitelist certain flows.

2.1.1 Learning Challenges

Using machine learning techniques to accurately detect anomalous network traffic is no easy feat. Sommer and Paxson [8] present a comprehensive analysis of the reasons for this difficulty and focus on five key points:

1. Machine learning models are generally best suited to make classification decisions based on previously seen data. That is, models provide good results when grouping together similar samples but are relatively poor at labelling never before seen data. Because not all potential attack strategies can be included in the training data, models will struggle in the face of new adversarial approaches.
2. The high cost associated with making incorrect classifications. While you may not cancel your subscription to Netflix because it recommended a show you ended up not liking, mistakenly labelling benign traffic as malicious or vice versa can have huge ramifications. In the context of our project, false negatives simply mean we've chosen not to whitelist an otherwise benign flow. These flows will then follow the normal path of being routed through an IPS, not helping solve the performance bottleneck problem but certainly not a crippling mistake. False positives can be more damaging, but we can mitigate these high costs by making our whitelists impermanent, causing mistakenly whitelisted flows to be reevaluated frequently as new data becomes available.
3. Traffic data is about as diverse a data source as one can find. Aggregating a large amount of real-world traffic data is a challenging task on its own, but determining if that data is actually representative or typical of the network is even harder. Thus, collecting quality training and test data is quite difficult. Because we have access to real traffic from Princeton's network, we believe we can assemble a quality set of both training and test data that encapsulates a significant portion of "normal" traffic behavior.
4. The "semantic gap" that exists in machine learning. This gap consists of the void between the collection of results and the subsequent execution of a correct responsive action. Put simply, once a classification is made, what should be done with that information? This gap varies from network to network and is very hard to generalize. What constitutes malicious traffic in one network may be typical of another. Machine learning models need to be able to understand or at least be aware of these subtleties. We have defined our problem very clearly as a binary classification problem. Any subsequent action taken after flows are classified is outside the scope of our project.
5. Evaluating the performance of these systems is also hard due to the lack of both real-world, large-scale deployments (ironically due to the aforementioned reasons) and access to quality training and test data. As a result, the quality of most systems is assessed in a simulated network environment, but as these simulations provide questionable realism, the evaluation process is fundamentally flawed. We can evaluate our system using real traffic data and the IPS as a benchmarking tool. Never whitelisting flows that the IPS flags as malicious would be a good foundational goal for our system.

2.1.2 Statistical Anomaly Detection

While not all possible attacker strategies can be known and incorporated into a detection model, certain well studied attack patterns can be effectively learned and thwarted. Sawaya, Kubota, and Miyake [7] combine a deep knowledge of attacker strategies with statistical flow analysis to produce a model that is quite effective at distinguishing legitimate and malicious traffic. The authors first note that, while individual packet inspection can be useful in gleaning information about attackers, it is not computationally practical to analyze each packet, which is why larger units (flows) are the preferred pieces of data to analyze.

Their approach focuses on three different types of attackers which they refer to as simple, obvious, and nuisance attackers. These designations relate to behaviors regarding sending traffic to specific closed ports. The authors assume that flows to these ports are probably malicious because legitimate hosts rarely use them. This is the foundational assumption of a learning phase that consists of generating a list of "weird" hosts, first by identifying closed TCP ports that receive a disproportionate number of individual SYN flags then by aggregating all flows destined for this port. These hosts constitute the set of obvious attackers. The characteristics of these hosts (number of flows/bytes/packets) are then used to generate a normalized vector feature set that can be generalized to identify other types of attackers. While these techniques are targeted rather narrowly towards specific attacker profiles, the results are impressive.

Using these methods, the authors were able to achieve a successful detection rate of about 90% with a false positive rate of only 7.1%. We can incorporate some of these attacker profiles into our classification system, but would have to first verify that these attacker profiles do not constitute benign traffic in the context of Princeton's network.

2.2 Leveraging Flow Data

Analyzing and processing individual packets is an arduous task both computationally and conceptually. Flows provide a slightly higher-level abstraction of this networking data type by grouping packets sharing similar characteristics together. Previous works have used flow data to increase visibility of the various behaviors on display in a chaotic network environment. We plan to assemble flows out of the raw traffic data produced by Princeton's IPS to learn which ones we can safely whitelist.

2.2.1 Flow Whitelisting Techniques

Flow whitelisting has been previously used in supervisory control and data acquisition (SCADA) networks to improve network security. Barbosa, Pras, and Sadre [2] present an approach in which flows aggregated over the course of a learning phase constitute the set to be whitelisted and any flow later observed that is not in this set triggers alarms. Flows that cause alarms can then be manually added to the set of whitelisted flows by an administrator. The authors make two fairly large assumptions with this approach: that the traffic data collected during the learning phase is entirely legitimate and that the learning phase is long enough to collect most legitimate flows. This strategy, however, is well suited to SCADA networks as a majority of the incoming and outgoing traffic is generated by automated procedures and is thus quite predictable.

Flows are constructed via the repeated application of four rules to group related sources of connections together: for TCP connections

the source is either the recipient of a SYN packet or the sender of a SYN/ACK packet, hosts using ports below 1024 are simply set as the source, previously observed protocol and port pairs are used to uniquely identify sources (hosts employing dynamic port allocation unsurprisingly trigger many alarms), and if none of the previous rules apply then the source is the destination of the connection's first received packet. Overall this approach performed well in the context of SCADA networks, but would definitely be difficult to generalize to larger networks with more a more varied traffic profile. However, some elements of this system could be incorporated into our project. Specifically, we could permanently whitelist certain administratively certified flows. Simply whitelisting all flows collected during a learning phase would not produce good results due to the diversity of Princeton's traffic.

2.2.2 Measuring Big Data Movement in Flows

Parallel TCP connections are used for large scientific dataset transfers to increase throughput. Thus, it is important to reconstruct parallel flow sets from traffic measurements. Addanki, Maji, and Veeraraghavan [1] studied these parallel flow sets to give system administrators a means of thinking about network planning, traffic engineering, and improving user performance. They had a four step approach: (i) reconstruct elephant flows from NetFlow records by identifying flows that sent more than 1GB in a 1-minute period within their lifetimes, (ii) determine the set of elephant flows to and from an Autonomous System from the elephant-flow data collected at each router, (iii) reconstruct parallel flow sets from the elephant-flows given specific identifiers, and (iv) characterize the size, throughput, and duration of flow sets. They presented a few findings in regards to: (i) number of parallel flow sets, (ii) size, rate, and duration characteristics, and (iii) comparison of flow set rates on the same paths. They noticed that popular values for the number of component flows in a parallel flow set were 2, 4, 8, and 16, which may be due to different processing powers at different labs and the amount of data they need to send. It also seems that duration is a huge factor on whether a user will choose the network or express shipping. Given that overnight delivery services are available, there were only a few flow sets lasted more than 10 hours. They also noticed that the rate variance is small for the parallel flow sets that use a larger number of component flows. By identifying these parallel flow sets, they were able to create an accurate picture of big-data movement. This work can be incorporated and expanded on in our project. By employing similar methodologies, we can characterize specific flow types (such as elephant flows) and, depending on the nature of the flow, either directly whitelist or use the containing flow group as an important feature value for later classification.

2.3 Detecting Malicious Traffic

Combining learning and data analysis techniques with flow aggregation technologies can be used to construct systems capable of identifying abnormal or adversarial traffic. Previous works have used NetFlow to collect traffic information for intrusion prevention purposes with varying degrees of success. We can build on these previous works to better leverage the NetFlow platform.

2.3.1 NetFlow Intrusion Prevention Systems

The Internet, a global network, is facing more and more issues in the realm of attacks and network security. Currently, there exists a defense strategy where all malicious flows are redirected to firewalls before hitting the network's Intrusion Detection System (IDS). However, this strategy may not satisfy the security needs of

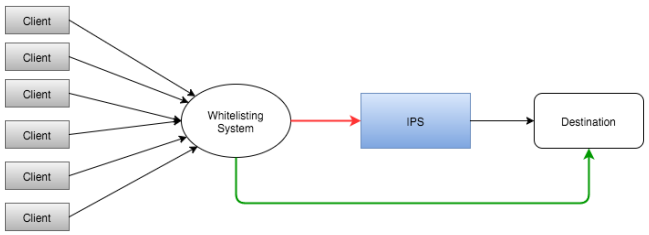

Figure 1. The potential workflow of a more mature version of the system described in this paper. Client flows are classified by the whitelisting system in real-time. Whitelisted flows are forwarded directly on to their final destinations whereas non-whitelisted flows are redirected through the IPS. The need for efficient classification is illustrated here, as slow classifications would make the whitelisting system the network's new bottleneck.

the network because the IDS produces a high rate of false alarms. Thus, Zhenqi and Xinyu [10] created a NetFlow-based anomaly traffic analysis system to detect attacks, identify intrusive behavior, and take decisive action against these malicious forces. The system has the following qualities: (i) it is easy to deploy NetFlow on every router in the internal network or on an external router, (ii) there is no need to have high-level information on NetFlow record because it allows the administrator to reduce the load needed to process the information, (iii) IDS based NetFlow anomaly analysis system has the ability to identify the newest attacks. The authors identify DOS, DDOS, and network virus traffic as the primary culprits of network intrusion. In order to warn the administrator of such network anomalies, the authors employ three tactics: (i) if the source IP of anomalous traffic can be identified, that connection can be severed (ii) when the user can pass the attacked port through an access control list, and (iii) in the event of knowing the destination of anomalous traffic, the user can use the static route option and set the destination address to null. The authors were able to successfully describe a system that implements NetFlow based intrusion detection system that can resist network attack with low cost and and not affect the backbone of the network performance.

2.3.2 NetFlow Analysis for Incident Detection

In the modern internet, incidents like DOS attacks, compromised machines, and policy violations are constant issues. Weigart, Hiltunen, and Fetzer [9] propose an approach for detecting these events by using NetFlow to analyze communication to/from a community of organizations (banks, governmental organizations, etc.), and alert the members of those organizations should suspicious activity be detected.

The authors claim that an attacker may target many organizations without being detected because such targeted attacks may not leave a large traffic footprint in the organization. This is because one machine with the desired control system may be enough for the attacker to achieve his/her goals. However, their system allows for the straightforward observation of communication behavior across multiple organizations in one community, so such mischievous behavior more readily comes to light. The system only looks at connections within the communities, between the communities, and they only consider external IP addresses that don't belong to any community.

A graph where the connections between nodes is then generated, indicating communications between communities or between an external IP address and a community. The edge weights between nodes are examined to quantify the importance of the communication (e.g., number of bytes sent, number of contacted individuals in the target organization, port number). As a result, compromised computers inside an organization are easily detectable. NetFlow is used to construct the sent messages using five fields: source IP, destination IP, source-port, destination-port, and transferred number of bytes. This allows the components to filtered and the system administrator to be alerted. One can also use a whitelist to eliminate any legitimate communication destinations from consideration (e.g., search engines, online retailers, etc).

2.3.3 Real-Time Intrusion Detection

NetFlow provides actionable information about the flow of data in and out of a network – but it doesn't do so quickly, slowing administrators' abilities to defend against sudden floods of malicious traffic. Hofsted, Bartos, Speretto, and Pras [3] present a few subtle modifications to the NetFlow architecture to enable this real-time detection functionality. Their efforts focus on reducing the timeout period between the collection of flow packets and the availability of that data for processing (currently at least 165 seconds).

The authors present a detection algorithm with flow measurements as input and a binary classification as output: benign or malicious traffic. The algorithm uses four key metrics to classify traffic: the rate of new flow record creation and the average duration, number of bytes, and number of packets per flow. This algorithm is then implemented as a NetFlow plugin which takes in flow measurements immediately after they are generated and passes its results to a filter. The filter can then blacklist flows tagged as malicious by either creating additional firewall rules or just ignoring packets with the same source IP as the malicious flow. Because the detection algorithm is a plugin for the flow monitoring application employed, the authors were able to reduce detection delays to 5 seconds without using an inordinate amount of additional compute resources (5% greater CPU usage and a 20 MB memory footprint in the worst case).

Overall, the authors demonstrated that their prototype could reliably and quickly identify DDoS attacks and they plan to port this detection system onto routers and switches in the future. We eventually hope to implement a real time version of our classifier that can provide tangible performance benefits for Princeton's network. A high-level picture of this system's workflow is presented in Figure 1.

3. Algorithm Design

In this section we discuss in detail the specifics of our data collection and classification workflow. While we primarily focus on the process leading up to our experimental evaluation, the steps outlined here could be further automated and improved for a real-world deployment of such a system, specifically pertaining to the assembly of the training data and the subsequent training of the classifiers.

3.1 Flow Processing

Our approach to flow collection and analysis is outlined below. The traffic in the Princeton network is captured, gathered, and aggregated by NetFlow. NetFlow, a feature introduced on Cisco routers, provides the ability to collect IP network traffic on an interface. These records are then concatenated into parallel flow sets to determine which communities are using parallel TCP connections. Using those parallel flow sets, we will start learning based on the time the flows were generated and source and destination IP. Once we are significant accuracy that can whitelist flows, we will start detecting the benign and malicious flows and take the appropriate actions.

3.1.1 Data Aggregation

In order to accurately determine whether or not to whitelist a given flow, we first needed to process the massive number of NetFlow records collected by Princeton's office of information and technology (OIT). These NetFlow records, collected from IP routers, provided detailed information involving packet counts, bytes, ports, timestamps, etc., on a per-flow basis. From this, we chose to reconstruct flows that encapsulate large data transfers, which we define as flows that send over 100, 250, or 500 MB in a 1-minute period within their lifetimes. From these size divided subsets, we reconstructed parallel flowsets, each consisting of multiple flows, from the single elephant flow characteristics identified by previous work [1]. We chose to identify parallel flowsets because the scientific community uses file transfer applications with file segmentation and reassembly features to better leverage parallel TCP connections in order to achieve high throughput and bandwidth. By identifying which groups are generating such high volumes of data, we can possibly try to avoid sending that information through the IPS, reducing the bottleneck and hopefully increasing the throughput of the entire network.

3.1.2 Parallel Flowset Detection

A flow k is represented using the following tuple:

$$\{s_k, d_k, x_k, y_k, z_k, f_k, l_k, o_k, r_k\}$$

where s_k: source IP address, d_k: destination IP address, x_k: source port number, y_k: destination port number, z_k: protocol number, f_k: UTC timestamp of the first packet of the flow seen, l_k: UTC timestamp of the last packet of the flow seen, o_k: cumulative number of bytes, and r_k: cumulative throughput. Using the algorithm identified by Addanki [1], we were be able to successfully identify high volume traffic. We looked at flows over a 1-week period. The first step was to identify all the flows in a certain day with the same source IP, destination IP, source port, destination port, and protocol number. After identifying those flows, it was important to group them based on timestamp. We looked for flows that start at approximately the same time (+/- τ). With that information, we then started looking at the cumulative size, rate, and duration to identify which flows need attention via the IPS and which flows could potentially bypass the system.

3.2 Traffic Classification

While machine learning techniques have been previously applied to the traffic classification problem, most works have focused on identifying harmful traffic [2][4][8][10]. Our application is subtly but crucially different: we adopt a malicious until proven innocent approach in which identification of benign flows is the priority. This simple change in perspective allows us to focus our classifiers' efforts on uncovering universally shared characteristics of non-threatening flows. A significantly easier task than inferring shared characteristics of all potential adversaries [9].

Name	Details	Type
src_ip	Source IP address.	Discrete
dst_ip	Destination IP address.	Discrete
src_port	Originating port of flow.	Continuous
dst_port	Destination port of flow.	Continuous
protocol	Flow protocol.	Discrete
packets	Number of packets exchanged.	Continuous
bytes	Number of bytes exchanged.	Continuous
duration	Total length of flow in seconds.	Continuous
pps	Packets per second.	Continuous
bps	Bytes per second.	Continuous

Table 1. Flow level feature set used to train models.

3.2.1 Problem Definition

We treated the problem as one of binary classification: flows will either be whitelisted or directed to the IPS for further inspection. While performing these flow classifications in real-time may be outside of the scope of this paper, the decisions we have made about our approach to this problem are motivated by the efficiency of model construction, classification, and deployment. Because we are employed supervised learning techniques to classify traffic, we leveraged the IPS to label our data. The IPS keeps detailed logs of all flows it flags as malicious. We treated this information as the ground truth, labelling any flow the IPS had deemed malicious with a 1 (don't whitelist, direct to IPS). All other flows were labelled with a 0 (whitelist, leapfrog the IPS).

While this approach may have resulted in an uneven distribution of positively and negatively labelled samples, we feel the size of our dataset mitigates this disparity. In this context, false positives and negatives vary hugely in potential consequences. False negatives won't have a noticeable effect on the system, as those flows will simply be directed to the IPS and any potential performance gains are forfeited. False positives, however, could be potentially quite damaging to the network, because flows that would have ordinarily been flagged and blocked by the IPS would instead be delivered to their destination normally, allowing potential adversaries to easily circumvent all of the IPS' defense mechanisms.

3.2.2 Feature Selection

Useful features for traffic classification can be derived from a number of different contexts. While previous works [11] have incorporated individual packet-based features into their models, as well as timing, entropy, and circumvention tool based features, our features are derived exclusively from high-level flow information because our dataset is primarily comprised of flows that have been preassembled by the IPS. Though this may seem like a detriment to our model construction, other works [10] [11] have produced classifiers that achieve high classification accuracies despite the use of similarly relatively limited feature sets. Additional features that we considered adding to this set include attack specific features (using previously known characteristics of well-documented attack types as features), context specific features (previously seen flows, time of day of flow beginning/end, IPs of established trusted sources etc.). This combination of both discrete and continuous features could also be easily transformed into a much larger set of Boolean valued features for furthered potential experimentation.

3.2.3 Classifiers

We hoped to try a variety of classifiers on our dataset, but ended up being constrained by the magnitude of our training data. We planned on initially focusing on four classification algorithms that are efficiently trained and work well with a smaller feature set: decision tree, k-nearest neighbors, multi-layered perceptron, and support vector machine. We planned on using Weka [3], a general purpose data-mining and machine learning toolkit written in Java, to carry out the brunt of the model construction and evaluation work.. As mentioned previously, our design decisions in this context are primarily motivated by a desire to minimize the latency of all aspects of the learning process. This low latency is important for two reasons. First, due to the diversity and sheer amount of traffic data, it is logical to retrain the classifiers quite often to account for traffic changes and potential adversarial advances. The size of these datasets thus dictates the choice of classifier. Secondly, our system is designed with deployment in mind, to operate somewhere within a network and rapidly make informed classification decisions in near real-time. Minimizing classification time will enable flows to be potentially whitelisted as soon as they are constructed by the IPS.

Our shift away from Weka and to Apache Spark (discussed below) forced us to reevaluate our classifier selection, as Spark has a much narrower selection of classifiers to choose from. As such, we decided to evaluate the performance of decision tree, random forest, and linear SVM models.

3.3 Experimental Evaluation

Our evaluation methodology can be divided into two distinct sections. The first consists of a detailed analysis of the accuracy of our various classifiers while the other will is an evaluation of the effects on the IPS of a hypothetical deployment of our whitelisting system. The two are closely linked, however, as poor classifier performance could still result in significant performance improvements (whitelist everything) but seriously compromise the security of the network.

3.3.1 Classifier Accuracy

We performed ten-fold cross validation on the various subsets of our flow data (partitioned into single day subsets) for each classifier. We tuned hyper-parameters to optimize classification accuracy and additionally hoped to evaluate the effects of removing/combining different subsets of our features on the true and false positive rates of the classifiers, though we were unable to conduct this experiment. High sensitivity was the primary goal of the classification process, due to the previously stated high cost of falsely whitelisting flows. After we optimized the performance of our models on our training data (while avoiding over fitting of course), we evaluated its performance on new data. The efficacy of our classifier was be judged by the size of the overlap between the set of whitelisted flows and the set of flows the IPS blocks.

3.3.2 Performance Impact

Measuring the potential savings of whitelisting traffic is quite straightforward. We used the single day data subsets for testing.

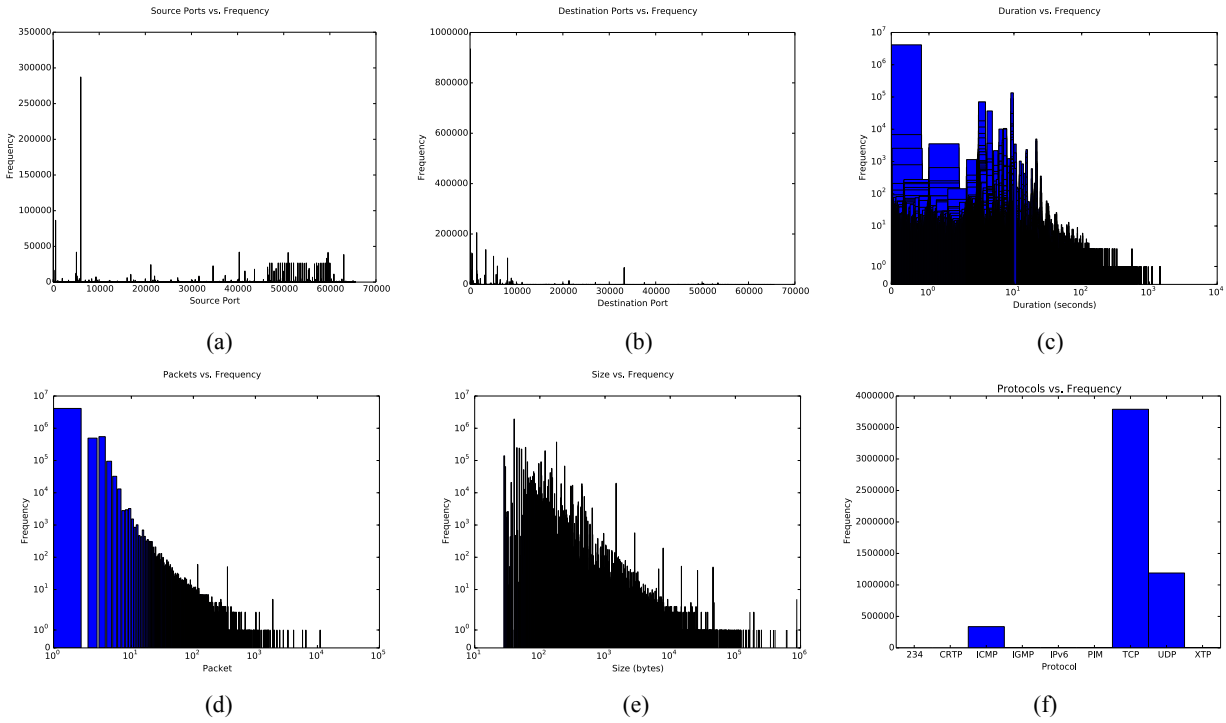

Figure 2. Frequency distributions of the various attributes of our aggregated flow data. (a) Source port distribution. (b) Destination port distribution. (c) Flow duration distribution. (d) Packet count distribution. (e) Byte count distribution. (f) Protocol distribution.

We used our classifiers to obtain the set of whitelisted flows for each of these data sets. Then, using the size of the data transfers of the whitelisted flows, it was easy to ascertain the reduction in load on the IPS. An important aspect of this evaluation was investigating the tradeoffs between the security and performance of the system by comparing the false positive rates of the classifiers with their respective data savings.

4. Results

In this section we present the results of our various experiments. We trained and tested our models on a combination of single-day subsets as well as on the full dataset. We additionally conducted experiments in which we varied the ratio of positively and negatively labelled samples in the training set. The experiments were conducted using the Apache Spark machine learning framework on both a Macbook Pro and on a number of compute nodes available via Princeton's Adroit cluster computing network.

4.1 Dataset

4.1.1 Dataset Profile

Getting access to the data we needed proved to be one of the most difficult aspects of this project and the data we did manage to obtain had some noticeable shortcomings, but still provided us a solid foundation on which to start building our models. The data we received was collected over the course of eight days from late October to early November by Princeton's IPS and amounted to 1.5 GB of raw flow data. After extracting feature values from these flows, we assembled 5.3 million samples to train and test with. The various frequency distributions of different flow attributes in our dataset are presented in Figure 2. Unfortunately, only 14 of these flows were flagged as malicious by the IPS and blocked. Because we were operating under the assumption that the IPS represented the ground truth (and was our only source of negative sample labeling), this deficit created a giant disparity between positively and negatively labeled samples in our dataset.

4.1.2 Performance Considerations

After having explored the usage of Weka for our project, we determined that it wasn't the best tool to handle the scale of network flow data. Additionally, we found that training and testing models on our laptops was also infeasible due to the high computational costs of our selected machine learning algorithms and techniques (holding the training samples in memory alone is an arduous task). Due to these factors, we decided to employ the Apache Spark [14] cluster computing framework. Spark allows massive amounts of data to be processed and queried across multiple compute nodes, making it uniquely well suited for machine learning tasks such as ours. Additionally, Spark includes a rather robust built-in library of machine learning algorithms and tools. While it does not boast the algorithm variety of Weka (as previously mentioned), it proved to be perfectly adequate for our investigations.

4.2 Classifier Performance

4.2.1 Hyperparameter Tuning

Before conducting classification experiments on all of our data, we first investigated the performance effects of tweaking the various parameters of our chosen classification algorithms. For decision

Dataset Information	Performance Metrics	Classifiers		
		Decision Tree	Random Forest	Linear SVM
Collected on 10/29/15 104,771 Flows	Accuracy	87.82%	77.71%	99.99%
	FPR	0.0%	3.33%	100%
	FNR	12.18%	12.18%	0.0%
Collected on 10/30/15 88,001 Flows	Accuracy	71.98%	76.74%	99.99%
	FPR	7%	2.86%	100%
	FNR	28.02%	23.26%	0.0%
Collected on 10/31/15 111,249 Flows	Accuracy	88.31%	85.41%	99.99%
	FPR	0.0%	4.5%	100%
	FNR	11.69%	14.59%	0.0%
Collected on 11/01/15 77,021 Flows	Accuracy	82.94%	71.7%	99.98%
	FPR	0.0%	3.33%	100%
	FNR	17.07%	28.3%	0.0%
Collected on 11/02/15 103,031 Flows	Accuracy	88.16%	84.91%	99.99%
	FPR	10.61%	19.95%	100%
	FNR	11.84%	15.09%	0.0%
Collected on 11/03/15 53,261 Flows	Accuracy	92.76%	87.2%	99.98%
	FPR	0.0%	17.5%	100%
	FNR	7.24%	12.8%	0.0%

Table 2. A comprehensive overview of the performance of our classifiers on per-day subsets of our dataset using classification accuracy and both false positive and negative rates as primary evaluation metrics. Decision trees appeared to provide the best combination of high accuracy and low false positive rates, with random forests performing similarly but noticeably worse in both categories. SVMs boasted the highest classification accuracy, but only because the constructed models elected to whitelist all flows, resulting in a very high false positive rate

trees, these consisted of information gain heuristics (entropy/gini impurity) and stopping conditions (maximum tree depth, minimum information gain for new nodes, etc.). We felt that a maximum tree depth of ten provided our trained model the flexibility to generalize to new data while avoiding over fitting and that the gini impurity heuristic provided much better performance in terms of both misclassification reduction and continuous feature handling. We made similar parameter decisions for our random forest models. Additionally, we set our number of trees to quite a low value, three, due to the diminishing performance gains of increasing this number and the large training time required of this ensemble method. For the linear SVM, the parameter options were limited so we simply limited the number of iterations to one-hundred in an effort to minimize training time.

4.2.2 Classifier Results

We first partitioned our data into per-day subsets to be used for the various experiments. Each of our experiments consisted of ten-fold cross-validation, splitting the data into 60% training and 40% test for each fold. The results from these experiments are presented in Table 2. In terms of raw classification accuracy, linear SVM models performed best, averaging 99.99% accuracy, followed by decision tree and random forest models, averaging 85.45% and 82.30% accuracy respectively. However, as previously discussed, the most important metric in effectively evaluating classifier performance in this context is the false positive rate.

Decision tree models did not label a single malicious flow as benign in any of our single day experiments, but had a false positive rate of 6.19% in our whole set experiment (discussed further below), bringing its total experimental average to 0.88%. This low false positive rate comes in stark contrast with the other evaluated models, as linear SVM and random forest models averaged 100% and 9.63% false positive rates respectively. Decision trees did however exhibit a significant false negative rate, averaging 14.54% across all experiments. The effects of falsely labelling benign flows as malicious while representing a considerable loss of potential performance improvements, are much more acceptable than the potentially calamitous effects of falsely labelling malicious flows as benign. The performance of the linear SVM is understandable because of the lack of negatively labelled samples in the dataset. In other contexts, the high accuracy provided by the SVM may even be desirable, although all it appeared to be doing in our experiments was labelling every flow as benign, a reflection of the general disparity between good and bad internet traffic.

In addition to the single day experiments, we also performed an experiment using a training set consisting of all of the days combined. Once again, the decision tree provided the best combination of sensitivity and accuracy.

Training Ratio	Performance Metrics	Classifiers		
		Decision Tree	Random Forest	Linear SVM
1:1	Accuracy	99.97%	96.92%	57.21%
	FPR	0.0%	0.0%	28.33%
	FNR	0.06%	6.1%	58.92%
2:1	Accuracy	93.84%	97.1%	52.39%
	FPR	0.0%	0.0%	0.0%
	FNR	10.29%	4.83%	79.49%
4:1	Accuracy	92.19%	95.18%	67.01%
	FPR	0.0%	0.0%	0.0%
	FNR	9.56%	5.9%	25.48%
Unscaled (~38000:1)	Accuracy	88.3%	90.92%	99.99%
	FPR	6.19%	15.95%	100%
	FNR	13.78%	28.3%	0.0%

Table 3. The effects of varying the ratio of positively to negatively labelled training samples on the performance of the various classifiers. Note that the scaled negatives were made entirely of the 14 originally flagged flows, greatly biasing the models against specific flow attribute values.

4.2.3 Malicious Flow Replication

Previous works [11] have investigated how varying the ratio of positively and negatively labelled samples in the training set effects the performance of the classifiers. We decided to apply this approach to our experiments by naïvely replicating our limited supply of negative samples to achieve more balanced ratios in our training data. Additionally, we conducted an experiment using the entire unfiltered dataset, including all of the flows from the individual days (~533,000 flows). The specific details of these experiments are the same as they were in the last section (10-fold cross validation, same hyperparameters, etc.). The results from these experiments are presented in Table 3. The accuracy of both decision tree and random forest classifiers appeared to increase with the reduction in sample label disparity, whereas the accuracy of the SVM decreased sharply. False positives also dropped to zero for all of the classifiers, but this wasn't very surprising considering the maliciousness of the original 14 flows was reinforced many times over due to the naïve scaling technique. The overall increase in the quality of the classifications, despite being somewhat artificially constructed, make us optimistic about applying similar techniques to more evenly balanced data sets in the future.

4.3 Bottleneck Reduction

After evaluating classifier performance, we turned to analyzing the performance ramifications of actually whitelisting the flows our classifiers labelled as benign. We repeated experiments similar to those previously discussed, but additionally calculated the total data size of the test set flows in bytes, as well as the total data size of the set of flows whitelisted by the classifiers. We divided the whitelisted data size by the total data size to ascertain the percent reduction in bytes that the IPS would have to process if our classifier was deployed on the Princeton network. The results of these experiments are presented in Figure 3. It is easy to see that

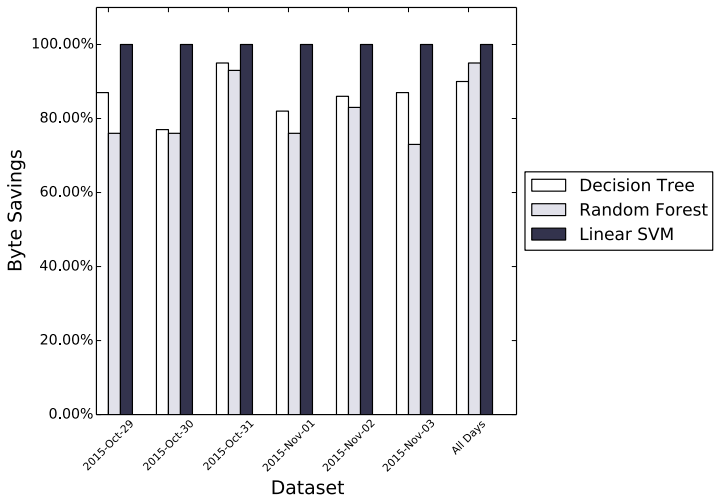

Figure 3. The percentage reduction of bytes that would potentially pass through the IPS in the case of a full deployment of our classifier. These bytes would instead leapfrog the IPS and be forwarded directly to their final destination.

the higher the false positive rate of the classifiers, the higher the potential performance gains. This phenomenon is most noticeably apparent in the linear SVM's performance, with the model directing no flows to the IPS due to its "whitelist everything" approach. Decision trees once again seemed to be the best option, providing a greater performance boost than random forests while also sporting a significantly lower rate of false positives.

While we were able to demonstrate the theoretical byte savings provided by our whitelisting system, the realistic gains may differ significantly. As we did not have full access to the IPS itself for our project (simply the data it collected), we were unable to assemble a clear picture of the extent of the created bottleneck, the per-flow analysis overhead, or the most performance hindering flow and data types.

5. Discussion

In this section to discuss our interpretations of the results of our various experiments, including the specific strengths and weaknesses of the classifiers themselves as well as shortcomings in our experimental approach. We also evaluate the most important aspects of this work to be expanded upon and further tested in the future.

5.1 System Efficacy

Despite the previously discussed shortcomings in the training data, we feel that the classifier results are quite promising. Decision trees seemed to provide a desirable combination of both high specificity and accuracy. As previously mentioned, the sensitivity of the classifiers is relatively unimportant in our context. These results are somewhat confounding, however, as the ensembles of decision trees provided by random forests should provide even better performance than the non-ensemble method. Perhaps more real-world, negatively labelled training samples and differently valued hyper-parameters could eliminate this discrepancy. Additionally, the limited feature set may have constrained the performance of the classifiers. It is unclear whether adding more flow-level features or

investigating the addition of novel packet-level features would have a greater effect on the sensitivity and accuracy of the classifiers.

5.2 Experimental Shortcomings

Over the course of conducting our investigations, we noted a number of key unsolved challenges that limited the scope of our contributions:

1. Limited access to IPS. While we are thankful that we were able to obtain a real-world dataset collected by the IPS, we really needed a greater degree of access to fine-tune our system to the subtleties of the device itself. We would have liked to measure the degree of bottlenecking that the IPS was responsible for (among other metrics such as throughput, etc.), but without information about the specific analysis routines the IPS performs and detailed timing metrics of these routines, it was impossible to glean a clear understanding of the performance constraints the device imposed upon Princeton's network. Additionally, more information about the specific rules used to flag flows as malicious would have helped us construct better features.
2. Improved training set. As mentioned previously, our training set exhibited a distinct lack of negatively labelled samples. While this is fairly representative of the normal distribution of good and bad traffic on the internet (there are far more innocent users than malicious ones), our classifiers really needed more negative samples to formulate a clearer picture of what malicious traffic typically looks like.
3. Lack of classifier timing evaluations. To reassert the effectiveness of our machine learning approach, we need to conduct experiments to determine the training, testing, and classification times of each of the classifiers. While training time is not of tantamount importance (periodically retraining the classifiers would be necessary to repel novel attack strategies), classification time is extremely important. High classification times means our system becomes the network's new bottleneck, taking that dubious title away from the IPS. Before we can think of deploying and testing the system in a real-world network environment, these metrics need to be recorded and minimized.

5.3 Future Work

There are a few aspects of our project that we would like to further explore, refine, and develop further. Most of these are aimed at eliminating the previously discussed shortcomings in our experimental execution. The first of these aspects is improving the quality of our data, which more specifically means aggregating data over a greater time period and increasing the number of negatively labelled samples used to train our classifiers. The merits of doing this are obvious and can only be achieved by increasing the period of data collection (as our access is real-time and not historical). Another aspect that we didn't fully explore in this project is the overhead of making classifications in real-time. While the classification speed was immaterial for the purposes of our experiments, it is vitally important to the successful deployment and operation of this system in a real network. Slow classifications simply move the bottleneck away from the IPS and to our classifier, not solving anything. In this same vein, optimizing the aggregation, training, and classification pipeline is another area of our project that is ripe for improvement. Our workflow was somewhat clumsy and involved repeatedly pivoting between multiple programming languages, learning frameworks, and various helper tools. Normalizing and optimizing this workflow would also be helpful in readying this system for a real world deployment, especially considering the likely need for frequent retraining of the classifiers.

5.4 Conclusion

In this work, we presented an approach for flow whitelisting to reduce the degree of bottlenecking within Princeton's network, hopefully leading to tangible performance benefits for both users and administrators. We demonstrated that classifiers trained with a relatively limited feature and dataset were capable of making high-quality predictions while providing comparable security guarantees to the IPS. Building on these results would preferably involve the implementation and deployment of a real-time classification system to divert traffic from the IPS.

6. REFERENCES

[1] Addanki, R., Maji, S., Veeraraghavan, M., & Tracy, C. 2015. A measurement-based study of big-data movement. In *Networks and Communications (EuCNC), 2015 European Conference on* (pp. 445-449). IEEE.

[2] Barbosa, R. et al. 2013. Flow whitelisting in SCADA networks. *International Journal of Critical Infrastructure Protection*. 6, 3-4 (2013), 150-158.

[3] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, *11*(1), 10-18.

[4] Hofstede, R., Bartos, V., Sperotto, A., & Pras, A. 2013. Towards real-time intrusion detection for NetFlow and IPFIX. In *Network and Service Management (CNSM), 2013 9th International Conference on* (pp. 227-234). IEEE.

[5] Intrusion prevention - Information Security: 2015.*http://www.princeton.edu/itsecurity/technical/ips/*. Accessed: 2015- 09- 30.

[6] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. 2003. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web* (pp. 640-651). ACM.

[7] KnowledgeBase - Office of Information Technology: 2015. *http://helpdesk.princeton.edu/outages/view.plx?ID=5070*. Accessed: 2015- 09- 30.

[8] Sawaya, Y., Kubota, A., & Miyake, Y. 2011. Detection of attackers in services using anomalous host behavior based on traffic flow statistics. In *Applications and the internet (SAINT), 2011 IEEE/IPSJ 11th international symposium on* (pp. 353-359). IEEE.

[9] Sommer, R., & Paxson, V. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on* (pp. 305-316). IEEE

[10] Soysal, M., & Schmidt, E. G. (2010). Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison.*Performance Evaluation*, *67*(6), 451-467.

[11] Thomas, K., Grier, C., Ma, J., Paxson, V., & Song, D. (2011, May). Design and evaluation of a real-time url spam filtering service. In *Security and Privacy (SP), 2011 IEEE Symposium on* (pp. 447-462). IEEE.

[12] Wang, L., Dyer, K. P., Akella, A., Ristenpart, T., & Shrimpton, T. (2015, October). Seeing through Network-Protocol Obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*(pp. 57-69). ACM.

[13] Weigert, S., Hiltunen, M. A., & Fetzer, C. 2011. Community-based Analysis of Netflow for Early Detection of Security Incidents. In *LISA*.

[14] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010, June). Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Vol. 10, p. 10).

[15] Zhenqi, Wang, and Wang Xinyu. "Netflow based intrusion detection system."*MultiMedia and Information Technology, 2008. MMIT'08. International Conference on*. IEEE, 2008