Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity

Blake Anderson
Cisco Systems, Inc.
blake.anderson@cisco.com

David McGrew
Cisco Systems, Inc.
mcgrew@cisco.com

ABSTRACT

The application of machine learning for the detection of malicious network traffic has been well researched over the past several decades; it is particularly appealing when the traffic is encrypted because traditional pattern-matching approaches cannot be used. Unfortunately, the promise of machine learning has been slow to materialize in the network security domain. In this paper, we highlight two primary reasons why this is the case: inaccurate ground truth and a highly non-stationary data distribution. To demonstrate and understand the effect that these pitfalls have on popular machine learning algorithms, we design and carry out experiments that show how six common algorithms perform when confronted with real network data.

With our experimental results, we identify the situations in which certain classes of algorithms underperform on the task of encrypted malware traffic classification. We offer concrete recommendations for practitioners given the real-world constraints outlined. From an algorithmic perspective, we find that the random forest ensemble method outperformed competing methods. More importantly, feature engineering was decisive; we found that iterating on the initial feature set, and including features suggested by domain experts, had a much greater impact on the performance of the classification system. For example, linear regression using the more expressive feature set easily outperformed the random forest method using a standard network traffic representation on all criteria considered. Our analysis is based on millions of TLS encrypted sessions collected over 12 months from a commercial malware sandbox and two geographically distinct, large enterprise networks.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; *Web protocol security*; • **Computing methodologies → Machine learning**;

KEYWORDS

TLS; Malware Detection; Network Security; Machine Learning

1 INTRODUCTION

Finding threats by passively monitoring network traffic has many advantages in terms of ease of deployment, and has been studied in industry as well as the academic literature [32, 35]. In addition to the passive monitoring requirement, accurately identifying malicious network traffic on an individual session-based level is necessary because it allows network devices, such as routers and switches, to efficiently enforce network policies. To further complicate a potential solution, the percentage of network traffic that makes use of encryption has been rapidly increasing, and, as expected, malware authors have also taken advantage of this trend to evade signature-based detection. Using a man-in-the-middle to decrypt the traffic and then using traditional technologies to detect threats is not ideal due to privacy concerns, and it is not always possible due to both legal and technical reasons.

Given the above set of constraints, machine learning on the encrypted network session's metadata is a natural solution. While not applied directly to detecting threats in encrypted traffic, this basic formula of machine learning and network metadata has been well-researched [6, 24, 31]. Unfortunately, these solutions have been slow to materialize as viable methods for real-world threat detection, and some critics have rightfully called into question the applicability of machine learning for this problem domain [25, 37].

Suitable false positive rates, while still maintaining high true positive rates on novel threats, has been difficult to achieve. In this paper, we highlight two primary reasons why this is the case: inaccurate ground truth and non-stationarity in network data. The most straightforward method to acquire labeled data for training is to use a sandbox environment to run malware and collect the sample's associated packet capture files for positively-labeled, malicious data, and to monitor a network and collect all connections for negatively-labeled, benign data. For the benign case, even after filtering the dataset using an IP blacklist [13], there will typically be a non-negligible percentage of network traffic that would be considered suspicious. For the malicious case, malware samples often perform connectivity checks, or other inherently benign activities. It is nearly impossible to identify all of these cases, and this must be taken into account when using supervised learning.

The second factor that affects classification accuracy is the non-stationary nature of network data. On a user level, the popularity of websites and services frequently change. For instance, transitioning your cloud hosted storage from `box.com` to `google.com/drive/` will have an impact on the encrypted traffic patterns observed. On a network and protocol level, change-points can be introduced when new protocols such as TLS 1.3 [34] or HTTP/2 [3] are released. These revisions can significantly affect

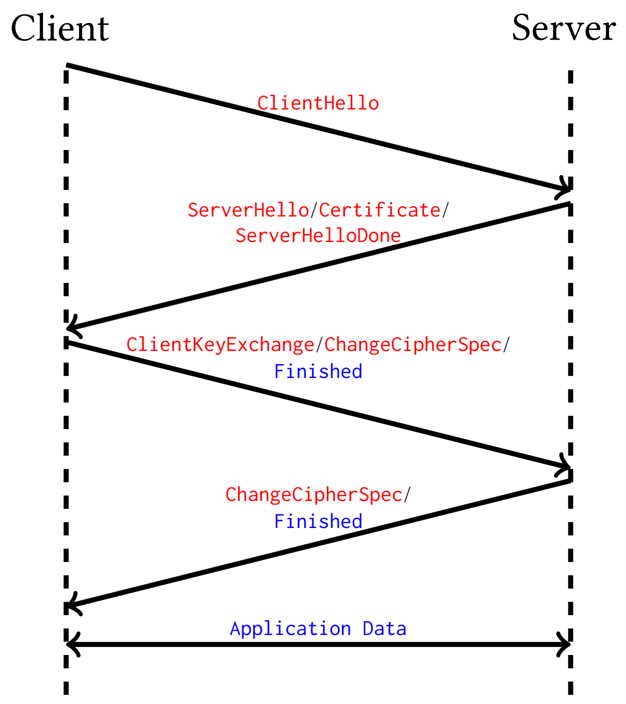


Figure 1: Graphical representation of a simplified TLS handshake and application data protocols. The feature sets used in this paper are taken from the unencrypted `ClientHello` message. Red text represents unencrypted messages, and blue text represents encrypted messages.

the structure of the handshake and application layer messages, impacting the data features used for classification.

We have designed and performed experiments that test how six common algorithms perform when confronted with inaccurate ground truth and an evolving data stream in the network security domain. We have found that some algorithms are more susceptible to the challenges that encrypted network traffic classification poses. For instance, we show that logistic regression's performance is not stable over time, and a standard support vector machine performs poorly in the presence of network traffic with corrupted labels.

In addition to examining several algorithms, we provide a comparison between a standard feature set used for this problem versus a custom feature set, which we developed with the help of domain experts. Iterating on the available features is often overlooked, but typically provides the most significant increase in performance [19]. Further confirming this line of thinking, we found that modifying the data collection process to generate the expert-guided, more expressive feature set significantly impacted the performance of all classifiers that were tested. For example, linear regression using the more expressive feature set easily outperformed the random forest method using a standard network connection representation [39] on all criteria that we considered. In general, pursuing a deeper understanding of the data and iterating on the data generation process proved to be extremely valuable.

It is also interesting to note that the additional data features can ease some of the burden with respect to accurate ground truth labeling. Malware typically performs connectivity checks by visiting a standard website, e.g., `https://www.google.com`. Using standard feature representations, this is impossible to differentiate from a benign client going to `https://www.google.com`. But, if additional features about the connection are included, such as the TLS handshake metadata, it becomes possible to distinguish these two cases because the TLS features provide information about the originating client.

Finally, given the real-world challenges posed by the network security domain and the efficacy we observed in our experiments, we offer concrete recommendations for practitioners in terms of algorithms and data features to correctly and robustly classify encrypted malware communication. Our analysis is based on millions of TLS encrypted sessions collected over 12 months from a commercial malware sandbox and two geographically distinct, large enterprise networks. We focus on the Transport Layer Security (TLS) protocol [18] due to its wide adoption.

The remainder of this paper is organized as follows: in Sections 2 and 3, we briefly review the TLS protocol and the algorithms used in our experiments. In Section 4 we outline our datasets and the feature representations that we extract from TLS encrypted traffic. Section 6 presents our results. Section 7 highlights related work, Section 8 reviews the reproducibility of our experiments, Section 9 considers the ethical aspects of our experiments, and Section 10 concludes.

2 BACKGROUND

TLS [18] is the primary protocol to secure many plain-text application protocols, e.g., HTTPS is the plain-text HTTP protocol over TLS. Figure 1 provides a graphical representation of a simple TLS session. The client initially sends a `ClientHello` message that provides the server with, among other fields, a list of cipher suites and a set of TLS extensions that the client supports. The cipher suite list is ordered by preference of the client, and each cipher suite defines a set of cryptographic algorithms needed for TLS to operate. The set of extensions provides additional information to the server that facilitates extended functionality, e.g., the Server Name Indication extension indicates the hostname of the server that the client is trying to connect to, which is important for virtual hosting. As explained in Section 4, all of the TLS data features used in this paper are taken from the unencrypted `ClientHello` message.

After the `ClientHello`, the server sends a `ServerHello` message that contains the selected cipher suite, selected from the client's offer list, which defines the set of cryptographic algorithms that will be used to secure the exchanged application data. The `ServerHello` message also contains a list of extensions that the server supports, where this list is a subset of what the client supports. At this time, the server also sends a `Certificate` message containing the server's certificate chain, which can be used to authenticate the server.

The client then sends a `ClientKeyExchange` message that establishes the premaster secret of the TLS session. Then the client and server exchange `ChangeCipherSpec` messages indicating that future messages will be encrypted with the negotiated cryptographic parameters. Finally, the client and server begin to exchange application data. In Figure 1, red text represents unencrypted messages, and blue text represents encrypted messages. The current TLS 1.2 handshake protocol provides a lot of interesting, unencrypted information. To enhance privacy, TLS 1.3 will be encrypting more of the handshake, e.g., the `Certificate` message will be encrypted, but the data features used in this paper will still be available. Many important details were omitted for the sake of brevity, but the associated RFC's provide the full specification [18, 34].

Because TLS encrypts many of the application-specific features, therefore making traditional deep packet inspection infeasible, many researchers have utilized side-channel information to make useful inferences on the TLS traffic [38]. These data features are typically constructed from the individual packet lengths and packet inter-arrival times of the encrypted session. Commonly used features include the mean of the packet lengths, *n*-gram or Markov chain based features derived from the sequence of packet lengths, or similarly constructed features for the timing information.

3 ALGORITHMS

We compared six common algorithms: linear regression, $l1$/$l2$-logistic regression, decision tree, random forest ensemble, support vector machine, and multi-layer perceptron. Our aim was to cover broad categories of different learning and optimization algorithms, not to create an individual hyper-optimized algorithm. For this reason, when tuning hyperparameters we simply used grid search and cross-validation over a set of standard values. We used implementations from Scikit-learn [33] for all algorithms except for the multi-layer perceptron, which used Keras [12]. And again, our focus in this paper is on supervised learning techniques [37].

3.1 Linear Regression

Linear regression is one of the simplest machine learning models. It finds a linear model such that the coefficient vector, $\mathbf{w}$, minimizes the residual sum of squares error between data samples and labels [7]. While not typically used for classification, the resulting hyperplane can be used to assign binary labels. Linear regression has the advantages of being efficient to train/test and the resulting predictions can be easily explained through the interactions between the weight vector and the data features. Some of the disadvantages of linear regression include being dependent on the scaling of the data features, inability to model nonlinear functions, and often performing poorly in a classification setting even when the data is linearly separable. This model was included as a baseline, naïve method.

3.2 Logistic Regression

Unlike linear regression, logistic regression is designed specifically for classification. Logistic regression returns a proper probability, which can be interpreted as the probability of a feature vector belonging to a specific class. We used two different versions of logistic regression: the first using $l2$-regularization and the second using $l1$-regularization [22]. Scikit-learn uses `liblinear` [21] and a coordinate descent algorithm [23] to train the classifiers. Similar to linear regression, the resulting models are easily interpretable because there is a one-to-one correspondence between the data features and the parameters of the model. Additionally, the $l1$-penalty produces sparse weight vectors, which further increases the interpretability of the model.

3.3 Decision Tree

It is often the case that the underlying function that is being modeled is not linear with respect to the input. The remaining algorithms in the section are all nonlinear. Decision Trees learn simple rules on the input features to partition the space into distinct classes [10]; Scikit-learn uses an optimized version of the CART algorithm. Decision trees are relatively efficient to learn and are easy to interpret, i.e., a set of rules can be associated with each output. Decision trees are robust against feature scaling, but are not robust against class skew; additionally, decision trees are susceptible to overfitting [20]. Decision trees can give a probabilistic outcome by reporting the fraction of data samples that belong to the same class in the leaf node.

We used grid search and cross-validation to adjust two tunable hyperparameters: the number of features to consider when looking for the best split and the maximum depth of the tree. For the number of features, we considered all features, and the square root and base-2 logarithm of all features. For the maximum depth, we considered no maximum, and the square root and base-2 logarithm of the number of samples.

3.4 Random Forest

A random forest uses an ensemble of decision trees to make predictions [9]. Each individual decision tree is learned from a bootstrap sample of the full dataset. In the Scikit-learn implementation, the output of the ensemble is the average probability taken over all individual trees. While the bias components of the individual trees are increased with this method, the variance of the averaged outputs is significantly reduced, which often results in superior performance. The random forest ensemble is less interpretable than a single decision tree, but can still provide a set of rules for trees that assigned high probability to an outcome.

Similar to the individual decision tree algorithm, we used grid search and cross-validation to adjust the number of features per split and the depth of the trees. We also adjusted the number of trees in the forest, considering values between 25 and 200 with a step size of 25. Validation performance typically plateaued between 100-150 trees for the various experiments.

3.5 Support Vector Machine

A kernel-based support vector machine learns a nonlinear function by using the *kernel trick* to project samples to a higher dimensional feature space where the different classes are more likely to be linearly separable [16]. These models are flexible with little bias given an appropriate kernel function, and are well suited for high-dimensional feature spaces. Support vector machine classifiers have been shown to be robust and produce optimal results on a wide range of problems [7]. These models typically have very low interpretability, but the kernel function between the target sample and the support vectors can be used to provide a list of similar samples in the training dataset.

For the support vector machine, we had the following tunable hyperparameters: the kernel function, the soft-margin parameter, and assuming a Gaussian kernel, the width of the kernel. We examined second and third degree polynomial kernels and a Gaussian kernel. For the soft-margin parameter and the width of the Gaussian kernel, we searched between 10^{-5} to 10^{5} in increments of an order of magnitude.

3.6 Multi-layer Perceptron

Multi-layer perceptron (MLP) models with two or more hidden layers have proven to be state-of-the-art for many tasks such as speech processing and image recognition [27]. These models have very little bias and can learn highly nonlinear functions. But, these models also have many parameters that need to be learned, and there is the potential for overfitting. Multi-layer perceptrons with one or more hidden layers suffer from a lack of interpretability, which is often important in the network security domain.

We again used grid search and cross validation to tune the hyperparameters of the MLP model. For the number of hidden layers, we searched between 2 and 5 with a step size of 1. For the number of neurons in each layer, we searched between 32 and 512 in increments of powers of two. Finally, for the dropout regularization parameter [26], we searched between .1 and .5 in increments of .05.

4 DATA

4.1 Collection Environments and Tools

We leveraged three environments for our data collection: two enterprise networks each with 500-1,000 active users and a malware analysis sandbox. For the enterprise networks, we passively monitored all outbound internet traffic in real-time. We used the HTTP `User-Agent` to determine that ~45% of the machines on these networks run Windows 7 and Windows 10, ~40% of the machines run OS X 10.x, and the remaining machines are either mobile devices, or run various flavors of Linux or Windows XP. Approximately 50% of the endpoints on these networks were unmanaged.

The malware analysis sandbox allows users to submit suspicious executables, and each submitted sample is allowed to run for 5 minutes. The full packet capture is collected and stored for each sample. Due to hardware constraints, the samples are only allowed to run for 5 minutes in either a Windows XP or Windows 7 (32-bit or 64-bit) based virtual machine. The user selects the operating system, with the default being Windows XP. ~75% of the malware traffic we collected used Windows XP, with the remaining samples using a variant of Windows 7.

These methods of data collection were straightforward, but could potentially lead to some biases in our experiments, which we now explicitly state. The two enterprise networks were located on distinct continents, but were both branches of the same company. So, while the network traffic will have many unique geographical characteristics, there will also be many overlapping sessions due to similar endpoint configurations. Because the malware sandbox restricts sample runtime to 5 minutes, any activity after this initial window will not be captured. Also, any malware sample that is not compatible with the selected Windows version or lacks a critical dependency will not run.

Time period	Enterprise	Malware
pre-May	620,072	208,368
May	616,823	15,316
June	596,848	8,832
July	619,859	18,836
August	553,164	13,429
September	545,931	21,114
September (Ent2)	735,195	N/A
Total	4,287,892	285,895

Table 1: The number of TLS encrypted sessions that we observed between August 2015 to September 2016; all month denotations in the table refer to 2016. Results are broken down by month. For the enterprise case, we began observing traffic in April 2016.

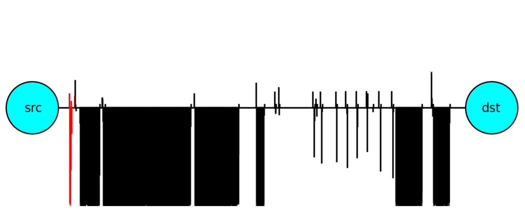
We wrote a `libpcap`-based open source package, Joy [30], to process the live traffic and the packet capture files. Joy converts the network data into a JSON format that contains all of the relevant data features, which made analyzing this data with standard tools trivial. For all experiments, we only used TLS sessions that completed the full TLS handshake, and sent application data. Joy anonymized all internal IP addresses for the enterprise traffic, and we did not retain the unprocessed, raw data.

4.2 Datasets

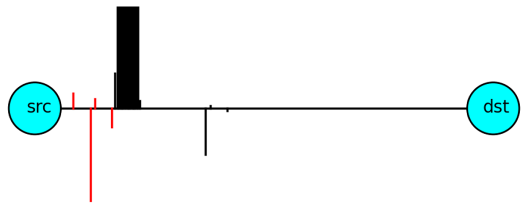
Table 1 provides a summary of the data that we collected, segmented by month. We began collecting the malicious pcap files beginning in August 2015, gathering ~10-30,000 new TLS sessions each month. We began monitoring one enterprise network, Ent1, in April 2016. We began monitoring a second, geographically distinct enterprise network, Ent2, in September 2016. We filtered the enterprise traffic using a popular blacklist [13], where the blacklist was updated daily. Despite this, some malicious sessions undoubtedly remained in the enterprise datasets. We further examine the ramifications of this in Section 6. Finally, for each enterprise network, we randomly sampled the data, without replacement, to create the final datasets.

5 DATA FEATURES

We introduce an additional axis in our experiments: the effect of a more expressive feature set developed by domain experts. Variations of a standard set of network traffic features combined with machine learning algorithms have been used in previous work [6, 39]. These features are typically derived from IPFIX [15] or NetFlow [14], and can be exported by traditional network devices. The enhanced features are less efficient to obtain, but could also be exported by network devices [29]. All features were normalized to have zero mean and unit variance. There are 22 and 319 data features in the standard and enhanced set, respectively.



(a) Google Search



(b) Bestafera

Figure 2: The TLS packet lengths and inter-arrival times for a typical Google search and malicious data exfiltration from `bestafera`. Upward and downward lines represent the sizes of packets being sent from client → server and server → client, respectively. The x-axis represents time.

5.1 Standard

For the standard set of features, we used features common in the literature, specifically the work presented by Williams et al. [39]. These 22 features included the minimum, mean, maximum, and standard deviation of:

- client → server packet lengths
- server → client packet lengths
- client → server packet inter-arrival times
- server → client packet inter-arrival times

The protocol, duration of the network connection, number of client → server packets and bytes, and number of server → client packets and bytes were also used. Williams et al. [39] provides a complete list of these features in their Appendix C: Table of Features. While most of these features are present in standard NetFlow records, the information about packet sizes is not. We included the features derived from the packet sizes in our experiments for the sake of consistency.

5.2 Enhanced

The enhanced feature set extends previous work by incorporating individual packet lengths and times, which gives a more detailed view of the behavioral profile of the application, and TLS metadata, which gives information about the library that the application is using for TLS. To provide the intuition that the domain experts used to determine which features were interesting in the context of TLS malware detection, `bestafera`, a particular malware sample known for keylogging and data exfiltration, is presented as we describe the enhanced features in depth.

5.2.1 Packet Lengths. Figure 2 shows the packet lengths and inter-arrival for two different TLS sessions: a Google search in Figure 2a and a `bestafera`-initiated connection in Figure 2b. The x-axis represents time, the upward lines represent the size of packets that are sent from the client to the server, and the downward lines represent the size of packets that are sent from the server to the client. The red lines represent unencrypted messages, and the black lines are the sizes of the encrypted `application_data` records.

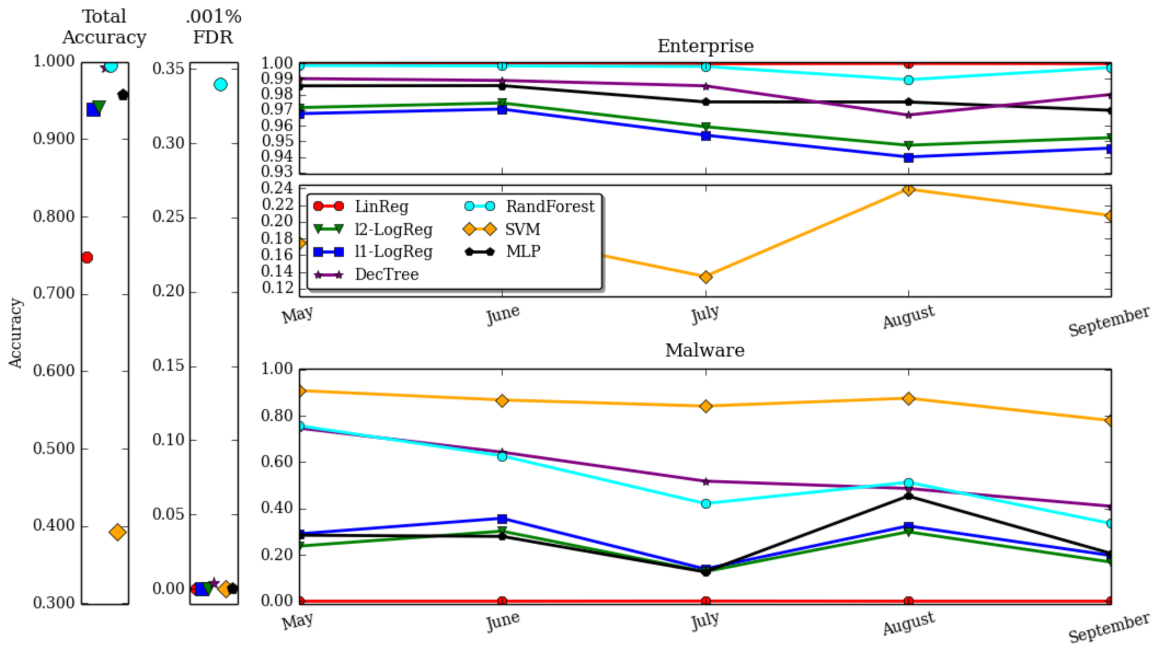
The Google search follows a typical pattern: the client's initial request is in a small outbound packet, followed by large response spanning many MTU-sized packets. The several alternating packets are due to Google attempting to auto-complete a search while the user was still typing. Once Google had sufficient confidence in the auto-completed result, it sent an updated set of results shown by another response spanning many MTU-sized packets. The server that `bestafera` communicated with began by sending a packet containing a self-signed certificate, which can be seen as the first downward, thin red line in Figure 2b. After the handshake, the client immediately begins exfiltrating data to the server. There was a pause, and then the server sent a regularly schedule command and control message. Packet lengths and inter-arrival times can't provide deep insight about the contents of a session, but they do facilitate inferences about the behavioral aspects of a session.

For the specific features, we record the sizes of the payloads for the first 50 packets of a session. We then represent these sizes as a first-order Markov chain. We assumed a 1,500 byte MTU, and created 10 states of 150 bytes each, and estimated the transition probabilities between states with the collected packet sizes.

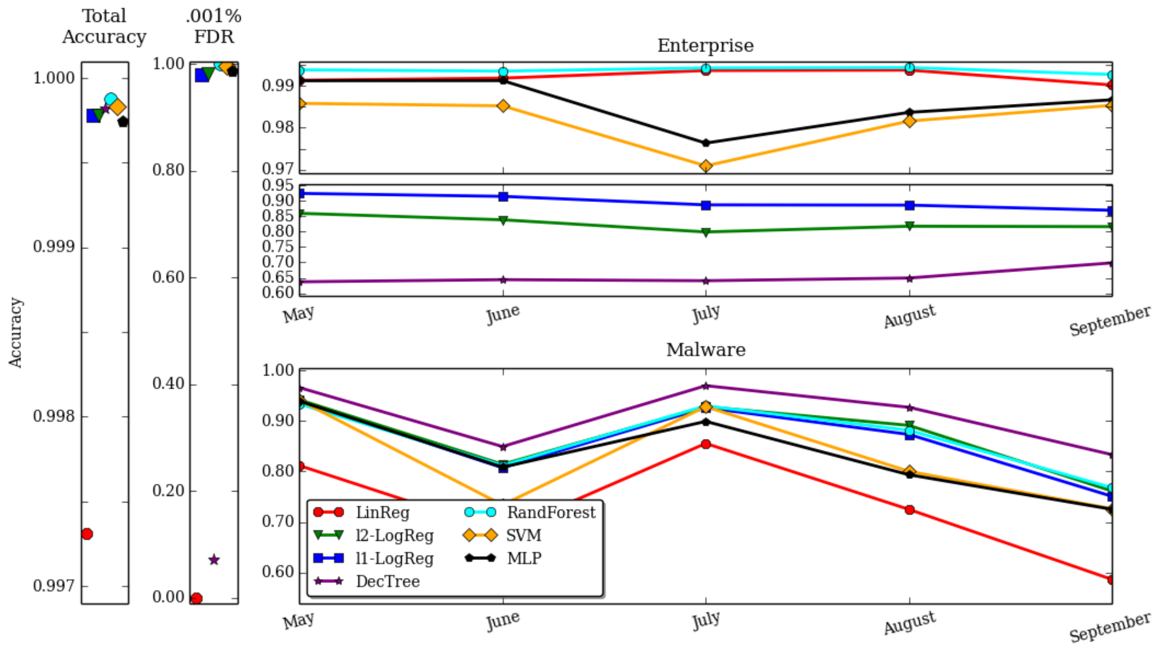
5.2.2 TLS Handshake Metadata. The TLS `ClientHello` message provides two particularly interesting pieces of information that can be used to distinguish different TLS libraries and applications. The client offers the server a list of suitable cipher suites ordered in the preference of the client. Each cipher suite defines a set of methods, such as the encryption algorithm and pseudorandom function, that will be needed to establish a connection and transmit data using TLS. The client can also advertise a set of TLS extensions that, among other things, can provide the server with parameters needed for the key exchange, e.g., `ec_point_formats` [8].

The cipher suite offer vectors can vary in both the number of unique cipher suites offered and the preferred components of each cipher suite. Similarly, the list of extensions varies based on the context of the connection. Because most applications typically have different priorities, these lists can and do contain a great deal of discriminatory information in practice. As an example, desktop browsers tend to favor heavier weight, more secure encryption algorithms, mobile applications favor more efficient encryption algorithms, and the default cipher suite offer vector of clients bundled with TLS libraries typically offer a wider range of cipher suites to help with testing server configurations.

Most user-level applications, and by extension a large number of TLS connections seen in the wild, use popular TLS libraries such as `BoringSSL` (`Chrome`), `NSS` (`Firefox`), or `SChannel` (`Internet Explorer`). These applications usually have unique TLS fingerprints because the developer will modify the defaults of the library to optimize their application. To be more explicit, the TLS fingerprint for the default `OpenSSL 1.0.1r` client, `s_client`, will



(a) Standard Representation



(b) Enhanced Representation

Figure 3: 10-fold cross validation accuracy, accuracy at a .001% false discovery rate, and the performance over time separated by enterprise and malware datasets.

most likely be different than an application that uses the `OpenSSL 1.0.1r` library to communicate. This is also why `bestafera`'s TLS fingerprint is both interesting and unique: it uses the default settings of `OpenSSL 1.0.1r` to create its TLS connections.

In this work, we derive features from the list of offered cipher suites and extensions. Again, these features are especially interesting because they provide information about the client that initiated the session, i.e., two otherwise identical sessions can be differentiated based on the client application. We observed 197 unique cipher suites and extensions in our data. We created a binary vector of length 197, and assigned a 1 to the appropriate entry if the TLS session supported the relevant cipher suite or extension.

6 RESULTS

Given the two main issues when performing supervised learning in the network security domain, non-stationarity in the data and noisy labels, we designed several experiments to assess the strengths and weaknesses of six popular algorithms. We also analyzed the effects that the enhanced feature set had on the algorithms.

6.1 Cross-Validated Accuracy

The left-most columns in Figures 3a and 3b show the 10-fold cross validation accuracy between the pre-May enterprise traffic and the malicious traffic. For the standard representation, a random forest ensemble and a single decision tree performed the best with a statistically significant margin determined by a 10-fold paired *t*-test at a 5% significance level. When the enhanced features are used, this discrepancy between classifiers disappears: all classifiers, with the exception of linear regression, have no statistically significant difference with respect to classification accuracy. While these results may be expected [40], we still consider them to be significant for the problem domain we consider.

Even moderately sized networks have tens-of-millions of network connections each day. Due to this scale, and the typical set of remediation actions that could be taken against a potentially malicious session, maintaining a low false positive rate is much more important than total accuracy. Figure 3 also reports the accuracy at a fixed .001%, or 1-in-100,000, false discovery rate (FDR). This is the accuracy of the classifier when it is only allowed one false positive for every 100,000 true positives. This strict measure was used as a proxy for results that should be expected in a real network setting, i.e., the relatively balanced class composition in the testing datasets is not realistic. The random forest ensemble was the only classifier to have a significantly greater than zero accuracy at a .001% FDR when using the standard representation. With the enhanced representation, all algorithms, with the exceptions of linear regression and the decision tree, performed the same using this metric.

6.2 Longitudinal Study

Threats and normal network behavior evolve over time, and a classifier that isn't continuously updated is at a natural disadvantage. In the case where updates cannot be deployed automatically, it is useful to understand how different algorithms will degrade over time. In this experiment, we trained all classifiers on the pre-May data, and Figure 3 shows the performance of these trained classifiers between May and September, 2016. The results for the enterprise data and the malware sandbox data are reported separately to identify weaknesses with respect to threat detection and because of the difference in scale of the respective datasets.

	Enterprise		Malware	
Algorithm	Standard	Enhanced	Standard	Enhanced
LinReg	99.92%	99.28%	0.00%	58.65%
l2-LogReg	93.35%	98.36%	16.86%	76.13%
l1-LogReg	92.75%	98.97%	19.71%	75.08%
DecTree	97.55%	97.02%	40.98%	83.33%
RandForest	99.53%	99.99%	33.54%	76.79%
SVM	11.94%	99.78%	77.98%	72.62%
MLP	95.90%	99.54%	20.61%	72.53%

Table 2: Classification accuracy on a validation dataset from a different network for both the standard and enhanced feature sets. Malware data from September is included to demonstrate majority-class classifiers.

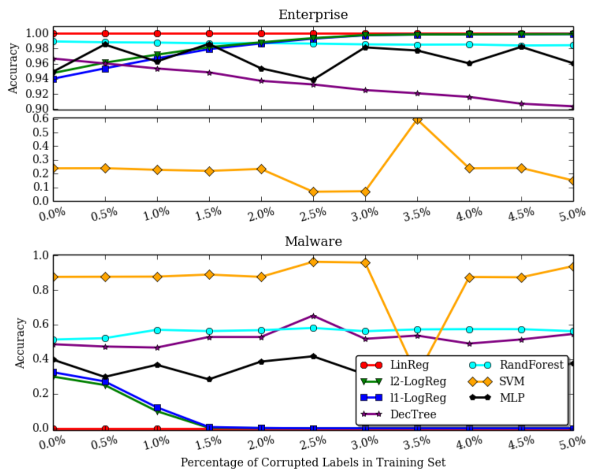
Using the standard representation, a random forest ensemble was clearly the best performing algorithm over time, maintaining its accuracy on the enterprise dataset. But, although the random forest still outperformed most algorithms on the malware dataset, its performance still fell significantly over time. There were also classifiers that were significantly biased towards one class, linear regression towards enterprise and support vector machine towards malware.

With respect to the enterprise data, the random forest using the enhanced representation almost exactly maintained its cross-validation accuracy across all 5 months. The random forest was also one of the most competitive algorithms when detecting malicious TLS sessions across all 5 months. It is interesting to note the robustness of linear regression when using the enhanced representation, and how is easily outperformed all other methods that used the standard representation.

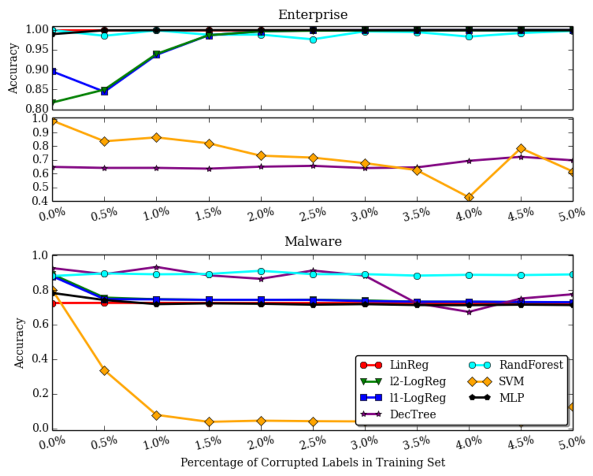
6.3 Distinct Network

We also tested how the classifiers performed when using a validation dataset collected from an enterprise network in a geographically distinct region. Traffic patterns are often different when considering distinct networks, especially when the networks are not located in the same country. For example, different governments suggest and/or mandate different cipher suites, which would impact the features we use for classification. Along these lines, it is important to know how well a trained classifier will generalize to a new environment.

Table 2 lists the accuracy of the various classifiers when tested on data collected at the new network. The classifiers were trained on the pre-May data. Similar to the previous results, we found that the random forest ensemble was extremely competitive, with four nines of accuracy. And again, the enhanced data features were more impactful than the choice of algorithm, e.g., linear regression with the enhanced feature set outperformed all other methods using the standard data representation.



(a) Standard Representation



(b) Enhanced Representation

Figure 4: The accuracy of the classification algorithms using the enhanced feature representation when a certain percentage of the labels are chosen at random and flipped.

6.4 Inaccurate Ground Truth

Noisy labels are a significant problem with real-world data collection in the network security domain. This is not only a problem when considering an adversarial setting, but also in the mundane setting of data preprocessing. A sandbox environment can generate many inherently benign network sessions, either from the malicious executable or the underlying operating system. Conversely, determining that a set of network connections from an enterprise network is truly benign is often impossible.

In this experiment, we randomly changed the class of a fixed percentage of samples in the pre-May training dataset, trained the classifiers, and then tested the classifiers on the September data collected from the original enterprise network and the malware sandbox. Figure 4 shows the results on both feature representations. For the enhanced data, the support vector machine was by far the least robust with respect to label noise. This has been independently observed, and although not addressed in this paper, solutions have been proposed [4].

Interestingly, logistic regression with both an $l1$ and $l2$ penalty was helped by the label noise. These methods were biased towards the malware samples, and introducing the label noise, at a rate of 1.5% to 5.0% significantly increased the classification accuracy on the enterprise dataset, and had no further degradation past 1.5% on the malware data. MLP and linear regression were both stable across all levels of label noise when using the enhanced data. Although linear regression's extreme bias led to a low accuracy at a fixed FDR, the bias made linear regression extremely robust to label noise. As Figure 4 shows, the random forest ensemble maintained its accuracy on the enterprise data despite the noisy labels. It was also significantly better on the malware data than all other classifiers with reasonable enterprise performance: maintaining ~89% accuracy versus ~72-74% accuracy. And again, the enhanced feature representation significantly helped the classifiers account for the noisy labels.

6.5 Adversarial Machine Learning

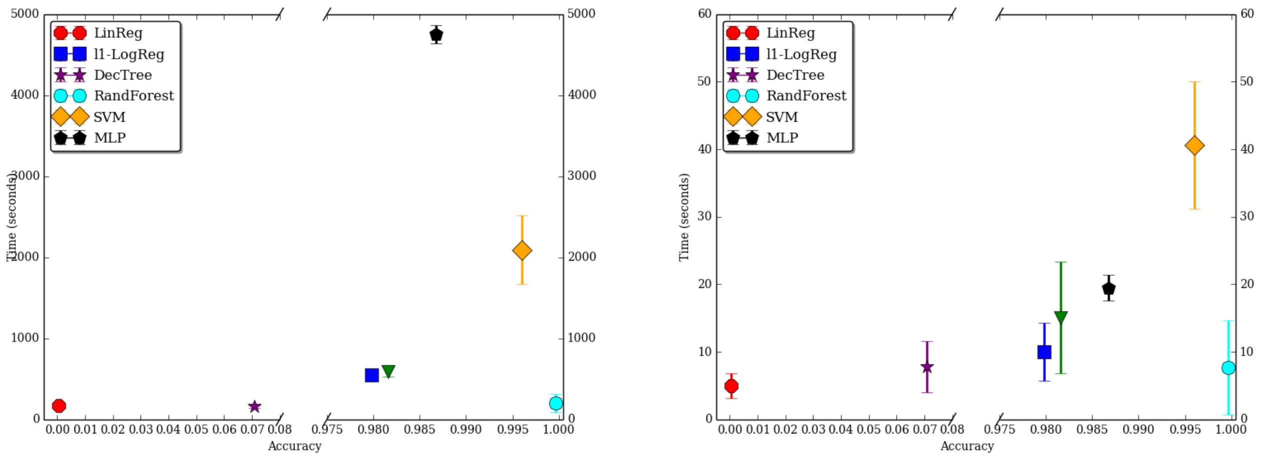
Adversarial machine learning promotes awareness of the types of attacks that can be perpetrated against a machine learning system, and attempts to provide solutions in the form of robust data features and training algorithms [5, 17, 36]. The goal of this current research is not to evaluate algorithms or data features in the adversarial setting, but to evaluate these parameters in the presence of noise that is expected in the network domain. That said, the previous results are also applicable to the situation in which an attacker has some influence over the training data, e.g., by submitting carefully crafted samples to the malware sandbox, or having an insider presence on an enterprise network where benign data is collected.

6.6 Computational Complexity

For the sake of completeness, Figure 5 plots the training and testing time of the classification algorithms in seconds versus the accuracy at a .001% FDR for the enhanced feature representation. These results align with what is typically reported for the respective algorithms. The MLP method [12] was slow to train, but quick to classify new samples. The random forest ensemble was quick to train and quick to classify new samples.

7 RELATED WORK

There has been previous work comparing supervised machine learning algorithms in the network domain [39]. In that work, Williams et al. evaluated the performance of naïve Bayes, C4.5, Bayesian Network, and naïve Bayes Tree algorithms on the problem of network application identification, i.e., HTTP vs FTP vs DNS, etc. They put forth the standard set of data features presented in Section 4. They found that the accuracy of all algorithms were similar, and the significant differentiator was the computational complexity of the algorithms. We build on this work by considering the encrypted traffic patterns of malware, a



(a) Training **(b) Testing**

Figure 5: The training and testing time of the classification algorithms using the enhanced feature in seconds versus the accuracy at a .001% FDR. In 5a and 5b, the error bars represent one standard deviation away from the mean time it took to train and test the models during the experiments of Section 6.1, respectively.

more extensive set of data features, and examining machine learning algorithms with respect to real-world problems: evolving data streams and inaccurate ground truth.

Label noise in the security domain has also been studied [28]. There has been research showing how to modify specific learning algorithms to account for label noise, e.g., adding a correction term to the kernel matrix of a support vector machine [4]. We add to this body of work by introducing a new set of results for the problem of encrypted, malware traffic classification. Additionally, we provide experimental results showing that additional domain-specific data features help to alleviate the negative impact of noisy labels.

The enhanced data features used have been studied in the security domain [1, 2]. However, these data features have not been studied in terms of the effects they have on different machine learning algorithms. Specifically, until this paper, there has been no experimental evidence showing that the unencrypted TLS metadata can reduce the effect that noisy labels and a non-stationary network environment have on machine learning algorithms.

8 REPRODUCIBILITY

Unfortunately, both the malware and enterprise data used in our experiments are highly confidential and cannot be publicly shared. Institution specific network data is notoriously difficult to share. With respect to the malware data, there do exist free malware sandboxes that allow packet capture downloads, but these offerings typically employ rate-limiting or do not have easy to script API's, and were therefore not suitable for our needs.

Those caveats aside, all of the data features used in this paper were generated by Joy, a network data collection and analysis tool that we open sourced and actively maintain [30]. Joy can be used to generate similar data when presented with malware packet captures files or is fed live traffic through a SPAN port at a monitoring point that sees Internet-bound traffic. For the machine learning algorithms, we used standard implementations and parameter settings from well known and supported open source projects, Scikit-learn [33] and Keras [12].

9 ETHICAL CONSIDERATIONS

All non-malware network traffic used in our experiments was collected and analyzed in accordance with the policies defined by our institution. This included the anonymization of all personally identifiable information such as internal IP addresses and usernames located in unencrypted HTTP sessions. Strict access control policies are maintained to ensure that all individuals accessing the data have the proper training to handle sensitive data, and have a valid justification to analyze the data.

It should also be noted that this research made no attempt to compromise the encryption of the studied TLS sessions. On the contrary, we are pursuing methods that can make useful, threat-relevant inferences on the encrypted traffic without the need to perform decryption, i.e., man-in-the-middle [11].

10 CONCLUSIONS

The network security domain poses a unique set of challenges, notably, the scale of the data, demand for very low false positive rates, evolving data streams, and noisy class labels. In this paper, we analyzed six common machine learning algorithms, and showed how they each performed on the problem of detecting malicious, encrypted network sessions. We specifically designed experiments to illustrate the algorithms' performance under the assumption of a typical operating environment, i.e., when the testing data is generated by a separate, but related, distribution and when the ground truth labels cannot be determined with 100% accuracy.

Although we found the random forest ensemble classifier to be the most robust for this problem domain, we demonstrated that the choice of features had a much greater effect on performance. The enhanced feature set was created by augmenting a standard feature set used in this domain with features identified by a domain expert and specifically tailored for encrypted network sessions. By not relying solely on features that were convenient to gather and engaging with domain experts to iterate on how the data would be best represented, all machine learning algorithms had significant improvements in performance. Combining diverse views of the data, such as features pertaining to how the application is transmitting data with features that are representative of the application, was the key innovation. As an example of the magnitude of the improvement, linear regression using the enhanced feature set easily outperformed the random forest ensemble using a standard network connection representation on all criteria considered.

11 ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their helpful comments. We would also like to thank the Cisco Technology Fund for funding this work, and Greg Akers for his support. This work was in part made possible by the help of Cisco CSIRT and ThreatGrid with respect to obtaining data, and Philip Perricone and Bill Hudson by their contributions to Joy.

REFERENCES

[1] Blake Anderson and David McGrew. 2016. Identifying Encrypted Malware Traffic with Contextual Flow Data. In *ACM Workshop on Artificial Intelligence and Security (AISec)*. 35–46.

[2] Blake Anderson, Subharthi Paul, and David McGrew. 2016. Deciphering Malware's Use of TLS (without Decryption). In *ArXiv e-prints*.

[3] Mike Belshe, Roberto Peon, and Martin Thomson. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (Proposed Standard). (2015). http://www.ietf.org/rfc/rfc7540.txt

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2011. Support Vector Machines Under Adversarial Label Noise. In *Asian Conference on Machine Learning*. 97–112.

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning Attacks against Support Vector Machines. In *International Conference on Machine Learning (ICML)*. 1807–1814.

[6] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: Detecting Botnet Command and Control Servers through Large-Scale NetFlow Analysis. In *ACM Annual Computer Security Applications Conference (ACSAC)*. 129–138.

[7] Christopher Bishop. 2006. Pattern Recognition. *Machine Learning* 128 (2006), 1–58.

[8] Simon Blake-Wilson, Nelson Bolyard, Vipul Gupta, Chris Hawk, and Bodo Moeller. 2006. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492 (Informational). (2006). http://www.ietf.org/rfc/rfc4492.txt

[9] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[10] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and Regression Trees*. CRC press.

[11] J Michael Butler. 2013. Finding Hidden Threats by Decrypting SSL. *SANS Institute* (2013).

[12] François Chollet. 2017. Keras. (2017). https://github.com/fchollet/keras Accessed: 2017-04-19.

[13] Cisco Talos. 2017. IP Blacklist Feed. (2017). http://www.talosintel.com/feeds/ip-filter.blf Accessed: 2017-04-19.

[14] Benoit Claise. 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational). (2004). http://www.ietf.org/rfc/rfc3954.txt

[15] Benoit Claise, Brian Trammell, and Paul Aitken. 2013. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011 (Proposed Standard). (2013). http://www.ietf.org/rfc/rfc7011.txt

[16] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[17] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, and others. 2004. Adversarial Classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 99–108.

[18] Tim Dierks and Eric Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). (2008). http://www.ietf.org/rfc/rfc5246.txt

[19] Pedro Domingos. 2012. A Few Useful Things to Know about Machine Learning. *Communications of the ACM* 55, 10 (2012), 78–87.

[20] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J Kay. 1997. A Comparative Analysis of Methods for Pruning Decision Trees. *Transactions on Pattern Analysis and Machine Intelligence* 19, 5 (1997), 476–491.

[21] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research* 9, Aug (2008), 1871–1874.

[22] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Springer.

[23] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. 2010. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software* 33, 1 (2010).

[24] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security Symposium*. 139–154.

[25] Matt Harrigan. 2016. Machine Learning is not the Answer to Better Network Security. (2016). https://techcrunch.com/2016/02/29/machine-learning-is-not-the-answer-to-better-network-security/ Accessed: 2017-04-19.

[26] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. In *ArXiv e-prints*.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444.

[28] Daniel Lowd and Christopher Meek. 2005. Adversarial Learning. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*. 641–647.

[29] David McGrew and Blake Anderson. 2016. Enhanced Telemetry for Encrypted Threat Analytics. In *IEEE ICNP Workshop on Machine Learning in Computer Networks (NetworkML)*. 1–6.

[30] David McGrew, Blake Anderson, Bill Hudson, and Philip Perricone. 2017. Joy. https://github.com/cisco/joy. (2017).

[31] Andrew W Moore and Denis Zuev. 2005. Internet Traffic Classification Using Bayesian Analysis Techniques. *SIGMETRICS Performance Evaluation Review* 33 (2005), 50–60.

[32] Vern Paxson. 1999. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks* 31, 23-24 (1999), 2435–2463.

[33] Fabian Pedregosa, Ga el Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[34] Eric Rescorla. 2017. The Transport Layer Security (TLS) Protocol Version 1.3 (draft 20). Intended Status: Standards Track. (2017). https://tools.ietf.org/html/draft-ietf-tls-tls13-20

[35] Martin Roesch. 1999. Snort - Lightweight Intrusion Detection for Networks. In *USENIX Large Installation System Administration Conference (LISA)*. 229–238.

[36] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. 2016. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1528–1540.

[37] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On using Machine Learning for Network Intrusion Detection. In *IEEE Symposium on Security and Privacy (S&P)*. 305–316.

[38] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. 2012. Botfinder: Finding Bots in Network Traffic without Deep Packet Inspection. In *ACM International Conference on Emerging Networking Experiments and Technologies (Co-NEXT)*. 349–360.

[39] Nigel Williams, Sebastian Zander, and Grenville Armitage. 2006. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *SIGCOMM Computer Communication Review* 36, 5 (2006), 5–16.

[40] David H Wolpert and William G Macready. 1997. No Free Lunch Theorems for Optimization. *Transactions on Evolutionary Computation* 1, 1 (1997), 67–82.