

To TTP or not to TTP?: Exploiting TTPs to Improve ML-based Malware Detection

Yashovardhan Sharma[†], Eleonora Giunchiglia[‡], Simon Birnbach[†], Ivan Martinovic[†]

[†]Department of Computer Science, University of Oxford

{yashovardhan.sharma, simon.birnbach, ivan.martinovic}@cs.ox.ac.uk

[‡]Institute of Logic and Computation, TU Wien

eleonora.giunchiglia@tuwien.ac.at

Abstract—In the last decade, machine learning (ML) methods have increasingly been applied to the task of malware detection. While these approaches have surely demonstrated their effectiveness, they still present limitations, some of which are a consequence of their purely data-driven nature. In this paper, we show how the MITRE ATT&CK framework of tactics, techniques, and procedures (TTPs) can be exploited to overcome such limitations and improve their ability to detect malware on networks. We conduct an extensive experimental analysis, testing 7 ML models on 5 large datasets comprising over 37 million flows. Our results clearly demonstrate that adding TTP-based features for training the models robustly improves their performance. Our models outperform the standard ones 922 times out of a total of 952, (i.e., 96.8% of the time), with the biggest improvements (up to 84.9% in terms of FPR) being observed in situations designed to be challenging for ML models.

Index Terms—Malware, Machine Learning, Intrusion Detection, MITRE ATT&CK, Networks

I. INTRODUCTION

In the last decade, machine learning (ML) models have increasingly been applied to the task of malware detection, and achieved considerable success. While these approaches have reached the point where it is remarkably hard if not impossible to improve their performance (see, e.g., [1], which reports a 100% F1-score for some test sets), they still present limitations (see, e.g., [2, 3]). These include the large amount and quality of data needed at training time, and the lack of robustness to adversarial attacks. Indeed, as stated in [2], standard ML models are only as good as the data used to train them, and while it is relatively easy to collect raw data from a network, it is much harder to get quality data which is (i) representative of the type of malicious attacks that are to be detected, (ii) properly labelled, maintained, and updated to keep track of the ever changing underlying relationships in the data, and (iii) used to initially train and/or update an ML model. The models’ strong dependency on the training data is also likely responsible for their high performance sensitivity when subjected to adversarial attacks, which are specifically crafted to fool the ML model into not recognising malicious activity. Ideally, we would like a model which (i) has consistently good performance, (ii) requires a small amount of training data, and (iii) is robust in the face of adversarial attacks.

In this paper, we demonstrate that it is possible to improve purely data-driven approaches to ML-based malware detection

on the network, by leveraging available background knowledge in the form of tactics, techniques, and procedures (TTPs) from the MITRE ATT&CK framework. We propose an approach that enhances the capabilities of any such system to detect malware using network traffic. In particular, (i) we automatically extract TTPs from raw network traffic captures, (ii) for each sample of network traffic, we create a Bag of Flows (BoF) representation of the network flows in the sample, (iii) we further annotate each sample with features about the TTPs detected in it, and finally (iv) we pass the BoF representation together with the TTP-based features as input to the preferred ML model(s) for malware detection. We expect the resulting ML models to have better performance, be less dependent on data, and be more robust to adversarial attacks, as a result of exploiting the domain-specific knowledge given by the TTPs.

To assess the effectiveness and robustness of our approach, we consider 7 popular ML models, and compare the performance of TTPxML models (i.e., the models utilising the TTP-based features) with the 7 standard ML models (i.e., models that do not use the TTP-based features) on the exact same parameters—flow-based features, samples, dataset splits, ML hyperparameters—with the only differentiator being the availability of the TTP-based features. We conduct an extensive experimental analysis involving a total of 1,551,535 malicious and benign samples, comprising of 37,981,348 flows. Beside the size, the datasets created for this paper represent an optimal playground to demonstrate the utility of TTPxML models over the standard ML models because: (i) the samples come from 5 different publicly available datasets, thus providing a broad range of malware samples, (ii) they have disjoint test sets, and (iii) they contain balanced target classes, which allows us to train the ML models under optimal conditions. Further, in order to measure the robustness of the approach, we considered various scenarios and 4 different metrics: True Positive Rate (TPR), False Positive Rate (FPR), F1-score and Accuracy. Our experimental analysis confirms that:

- 1) The TTPxML models consistently outperform the standard ML models. Indeed, throughout the course of our experiments, we compare the models a total of 952 times, and the TTPxML models outperform the standard models 922 times (i.e., 96.8% of the time) and in the 30 cases in which TTPxML has worse performance, the

ML Method	Work	Features	Target Class Domain
Log. Regression	Bapat et al. [4]	Global statistic of sample (e.g., mean / SD of flow duration)	Malware
AdaBoost	Khan et al. [5]	URL	Malware
Decision Tree	Wang et al. [6] Bekerman [7]	HTTP request features and TCP flow features Features from protocols: HTTP, SSL, DNS, TCP, IP, UDP	Malware + Family Malware + Family
SVM	Bartos et al. [8] Huang et al. [9] Tellenbach et al. [10]	Vectorial representation of sample URL Entropy of network flow features	Malware Sample Phishing Anomaly + Anomaly Type
Random Forest	Soska et al. [11] Bilge et al. [12] Bekerman [7]	Content of Web pages Flow size, time Features from protocols: HTTP, SSL, DNS, TCP, IP, UDP	Infected Websites Botnets Malware + Family
DNN	Piskozub et al. [13] Feng et al. [14] Do et al. [15] Cordero et al. [16]	Network flow features Static features + Meta information for each packet Entropy of network flow features Entropy of network flow features	Malicious Flows + Malware Family Malware + Category + Family Benign, DDoS, PortScan, P2P DoS, Net Scan Anomaly
XGBoost	Dhaliwal et al. [17] Bhattacharya et al. [18]	Network-level + Host-level features Network-level + Host-level features	Anomaly Attack Type
All of the above	Ours	Vectorial representation of sample + TTP-based features	Malware

TABLE I: Overview of the existing state-of-the-art approaches focusing on classification of malicious traffic on networks.

differences are not statistically significant in 29 cases.

- 2) TTP-based features are particularly helpful in challenging scenarios, where data is either scarce, or flooded with benign traffic generated by an adversary trying to camouflage their activity. In such cases, we report an improvement of up to 19.8% for F1-score, 15.9% for accuracy, 28.4% for TPR and 84.9% for FPR.

The rest of the paper is organised as follows: in Section II we give an overview of the MITRE ATT&CK framework and its applications, in Section III we survey existing work on malware detection on networks using ML techniques, and in Section IV we describe our proposed solution. We then conduct a thorough evaluation of our approach in Section V. Finally, we discuss the findings from our experiments in Section VI, and provide conclusions in Section VII.

II. BACKGROUND

The MITRE ATT&CK framework [19] is a globally accessible knowledge base of post-compromise adversarial tactics and techniques, based on real-world observations. It represents a behavioural model that is described by a wide-variety of adversary tactics, techniques, and procedures. More specifically, within the context of ATT&CK:

- A *tactic* signifies a tactical goal of the adversary. It is the reason why the adversary performs any action. *e.g.*, Credential Access.
- A *technique* represents the method by which an adversary achieves a tactical goal. A *sub-technique* within a technique corresponds to a more specific action within the given method. *e.g.*, T1003 - OS Credential Dumping.
- A *procedure* is a specific implementation of the method the adversary uses to fulfil a tactical goal. *e.g.*, LaZagne¹.

The ATT&CK framework has been used extensively in the literature to tackle various challenges within a cyber security context. For instance, it has been used in research

(see *e.g.*, [20, 21, 22, 23]) to track the rapid growth of novel malware and understand the nature of threats posed by them. Similarly, it has been used as a means of extracting structured information (such as TTPs) from large volumes of Cyber Threat Intelligence (CTI) reports, and converting them into actionable knowledge that can be exploited by analysts for cyber defence (see *e.g.*, [24, 25, 26]). Finally, the MITRE ATT&CK framework has also been utilised as part of a larger pipeline to help with detection tasks, such as when systems incorporate knowledge from TTPs to detect malicious behaviour (see *e.g.*, [27, 28, 29, 30]).

III. RELATED WORK

Traditionally, malware detection on networks has been performed by developing either heuristics-based or rule-based methods, (see *e.g.*, [31]), which are not able to scale with changes in traffic patterns or attacks and do not take advantage of the high volumes of data available today. To overcome such limitations, ML methods have been increasingly applied to the task of malware detection on networks. However, contrary to our proposed approach, such methods do not explicitly deploy the extensive domain knowledge that has been gathered by cyber security experts over the years, to create more informative features. Instead such knowledge is mostly used to either choose the most suitable ML model for the task, or to select/transform the set of features in the most suitable way.

An example of research that uses domain knowledge to create better feature representations is the work by Bartos et al. [8], where the authors create vectorial representations of each network sample such that they are invariant to: (i) the scaling and shifting of the flow-level feature values and, (ii) the permutation of the flows in the sample. On the other hand, research that focuses on anomaly detection, such as [10, 15, 16], computes the entropy of each feature and uses them to create a representation that is passed on to ML models.

Other research simply uses the domain knowledge to select the best subset of features for the task at hand. Among these,

¹<https://github.com/AlessandroZ/LaZagne>

Field	Property
Flow Hash	The hash associated with each unique flow.
Sample Hash	Hash of the sample corresponding to the flow.
Unique ID	A unique ID associated with each flow.
Dataset	The dataset to which a flow belongs.
Source Port	The port from which the flow originates.
Destination Port	The port on which the flow connects to.
Start time, end time	Flow start or end time.
Duration	Flow duration in seconds.
Protocol	IP protocol identifier in decimal format.
Entropy	The Shannon Entropy for the flow payload.
Applabel	The application label, as identified by YAF.
Source IP, Dest. IP	Source and destination IPv4/IPv6 address.
Type, Code	ICMP type or code in decimal format.
Isn, Rsn	Forward or reverse TCP sequence number.
Flags	4 properties representing various TCP flags.
Tag, Rtag	802.1q VLAN tag in forward/reverse direction.
Pkt, Rpkt	No. of packets in forward/reverse direction.
Oct, Roct	No. of bytes in forward/reverse direction.
RTT	Round-trip time estimate in milliseconds.
End-reason	Reason for termination of flow.

TABLE II: Flow properties.

one such example is the work by Bekerman et al. [7], in which the authors select 972 behavioural features across all network layers and three different protocols (DNS, HTTP, and SSL). Feng et al. [14] on the other hand, argue for the exploitation of both static features and network level features, since attackers nowadays use code obfuscation and repackaging techniques to conceal their behaviour and evade detection.

Finally, with the advent of deep learning, many works now use the available domain knowledge to choose the most suitable topology of neural networks to apply to the task. For instance, Piskozub et al. [13] try to distinguish benign and malicious flows on a network, and, if the flow is malicious, they then try to classify the type and family of the malware. Given the extensive literature on the topic of malware detection on networks using ML models, we give a summary of it in Table I, while for a more generic overview of the methods that have been developed for malware detection and classification we refer to the surveys [2, 3, 32, 33].

IV. METHODOLOGY

Our methodology consists of four basic steps:

- 1) We transform raw network captures exhibited by samples into privacy-preserving NetFlows.
- 2) We annotate each flow with all the TTPs that it matches.
- 3) For all flows corresponding to a particular sample, we then create a single Bag of Flows (BoF) which is a vectorial representation of the sample.
- 4) We then employ a variety of binary classifiers to decide whether a given sample is benign or malicious.

Below we describe each step in detail.

A. Network Flows

Raw network traffic data is often captured and analysed as PCAP files using applications such as Wireshark and tcpdump. While this format has several advantages, we instead opt to transform these network captures into NetFlows (or simply, flows) as they are: (i) lightweight, and thus easy to process,

Tactics	Techniques
Reconnaissance	T1590 - Gathering Victim Network Information
	T1595 - Active Scanning
Credential Access	T1557.001 - Man-in-the-Middle
	T1046 - Network Service Scanning
Discovery	T1124 - System Time Discovery
	T1135 - Network Share Discovery
Lateral Movement	T1021.001 - Remote Services (RDP)
	T1021.004 - Remote Services (SSH)
	T1550.003 - Use Alternate Authentication Material
	T1563.001 - Remote Service Session Hijacking (RDP)
	T1563.002 - Remote Service Session Hijacking (SSH)
	T1570 - Lateral Tool Transfer
Command and Control	T1071 - Application Layer Protocol
	T1090 - Proxy
	T1105 - Ingress Tool Transfer
	T1571 - Non-Standard Port
Execution	T1053 - Scheduled Task/Job

TABLE III: List of supported TTPs from the MITRE ATT&CK framework.

(ii) small in size, and thus require less storage space, and (iii) privacy-preserving, since they use only metadata and are compatible with encrypted traffic. In particular, we use a specific type of NetFlow, Yet Another Flowmeter (YAF) [34], which is designed for high performance and scalability across large networks, while also balancing the need for capturing information and maintaining privacy on the network. Finally, each flow is annotated with the various features describing its properties, which can be seen in Table II.

B. TTP Annotation of Flows

We select 15 TTPs from the MITRE ATT&CK framework which are specifically suited to NetFlow traffic; a complete list can be found in Table III. These encompass 11 tactics, 9 techniques, and 6 sub-techniques, which represent a sufficiently diverse set of TTPs to capture a wide variety of network-based malware. In order to obtain TTP labels for our network datasets, we follow a similar approach to that proposed in [29], and create a mapping between specific flow properties and TTPs. For instance, in order to detect the technique T1124 (System Time Discovery) we consider one specific procedure that is used for discovering time on the network, namely the Network Time Protocol (NTP). Since NTP uses a specific port, 123, to send and receive time synchronisation messages, we are able to map any TCP/UDP flows that communicate using that port to the technique T1124.

C. Bag of Flows

To create the BoF representation of a sample, we first create a vectorial representation of each flow. To do so, we transform each non-numerical feature to a numerical one. We transform the categorical features, i.e., “End-reason”, “Protocol” and the 4 features representing TCP flags, into one-hot encoded vectors. “Isn” and “Rsn”, though categorical, have too many unique values (63,796 and 8,229,092, respectively) and thus we drop them. We then consider the feature “Destination IP”, which we map into 32-bit integers retaining the ordinal information inherent of IP addresses. We drop the features “Flow Hash”, “Sample Hash” and “Unique ID” as they are identifiers, together with the feature “Dataset” as it is a spurious indicator of maliciousness. We also drop the features

“Source IP”, “Start Time” and “End Time” since they capture information that is not useful when executing in a sandbox.

Once we have the per-flow vectorial representations, we need to aggregate them. Many different methods have been proposed in the literature for such aggregation. We opt for a variation of the Bag of Words model [35] model proposed in the natural language processing field, which we call Bag of Flows (BoF). While we propose our own version of BoF, there are other methods called BoF in the literature. For example, Divakaran et al. [36] define a BoF as a set of flows localised in time, and sharing the same destination IP address, destination port, and protocol. Then, they classify each flow and reach a decision regarding the maliciousness of the BoF by majority voting. Zhang et al. [37] use the term BoF in a similar way, and once a BoF has been identified, a Naive Bayes model classifies each flow, and different aggregation policies (e.g., sum rule) are tested to classify the entire BoFs. Their definition of BoF is thus closer to our definition of a network sample. On the other hand, similar to our approach, Bartos et al. [8] use the term BoF to indicate a vectorial representation of a network sample. Contrarily to our approach however, Bartos et al. assume that each sample must have at least 5 flows (to be able to compute a meaningful representation) and a maximum of 100 (to ensure that the computational cost is controlled). Indeed, we do not put any upper or lower bound on the number of flows per sample. To create our BoFs, we perform the following steps:

- We apply min-max scaling to each flow-level feature.
- We transform each flow-level feature in a normalised histogram (i.e., a histogram recording the proportion of cases that fall into each category). Since not all the features are categorical, for each continuous feature we define B bins, and for each bin we record the proportion of flows in the sample having value belonging to that bin. In our experiments, we set $B = 5$.
- For each TTP we create two aggregate features: one representing the proportion of flows in the sample that match the given TTP, and the other representing the total number of flows matching the given TTP (this feature is then normalised over all the samples).
- We gather global features about all the TTPs, namely: (i) the number of unique TTPs matched by the sample, (ii) the total number of TTPs matched by the sample, (iii) the number of flows matching at least one TTP in the sample, (iv) the proportion of flows matching at least one TTP in the sample, (v) the mean of the number of TTPs matched by the sample, and (vi) the standard deviation of the number of TTPs matched by the sample. All these features are then normalised over all the samples.
- Finally, we concatenate the three representations obtained above and construct a single vector.

This final vector represents our BoF for a given sample, and is subsequently passed on to a ML classifier.

D. Classification

To test our hypothesis that TTPs help improve the performance of ML models, we consider the ones commonly used to

Dataset	Samples	Flows
Ember Malicious [38]	435,741	26,567,527
MalRec [39]	37,763	12,273,502
MalShare [40]	1,268,923	32,310,907
VirusShare [41]	595,098	12,217,493
Ember Benign [38]	155,432	1,423,023
Total (unique)	2,131,832	83,371,480

TABLE IV: Datasets statistics.

detect malware on networks, as shown in Table I. In particular, we test: (i) Adaptive Boosting (AdaBoost) [42], (ii) Decision Tree [43], (iii) Logistic Regression [44], (iv) Deep Neural Networks (DNN) [45], (v) Random Forest [46], (vi) EXtreme Gradient Boosting (XGBoost) and (vii) Support Vector Machine (SVM) [47]. For this analysis, we implemented the DNN as a feed-forward neural network with ReLU non-linearity and four hidden layers.

V. EVALUATION

The goal of our evaluation is to determine whether the TTP-based features are actually robustly helping the various kinds of ML models to discern between malicious and benign samples. Thus, in designing the experiment, we start considering a varied population of malware samples from the 5 datasets in Table IV, four of which contain malware samples while only one, namely “Ember Benign”, contains benign samples. Then, to be able to compute the average and standard deviation, we create 5 different datasets sampling from the original datasets in Table IV, ensuring that they have (i) balanced target classes so that we can train the ML model under ideal conditions, (ii) disjoint test sets, (iii) as many samples as possible given the previous two requirements, and (iv) malicious samples randomly sampled from the 4 malicious datasets. Finally, to check the robustness of the approach, we considered 3 experiments in which we vary:

- 1) the size of the dataset used for training,
- 2) the bin size B used to create a bag of flows, and
- 3) the percentage of benign flows in each malicious flows.

The first experiment simulates the case where there is scarcity of data, e.g., in the case of novel attacks where few samples are initially available. The second experiment analyses how dependent the results are on the chosen vectorial representation of the samples. The third experiment considers an adversarial scenario, where an adversary attempts to camouflage their malicious activity using varying amounts of benign traffic.

For each experiment, we report the average and standard deviation of the seven models in terms of F1-score, accuracy, FPR and TPR. Due to limited space, we only plot the results for F1-score, FPR and TPR, as accuracy follows the trends of the other metrics, and we present its results in the text.

A. Robustness Under Data Scarcity

A key limitation of ML systems is that they are data greedy. We argue that TTP-based features can be especially helpful in assisting ML-based intrusion detection systems (IDS) in data scarce situations. This is particularly desirable because very

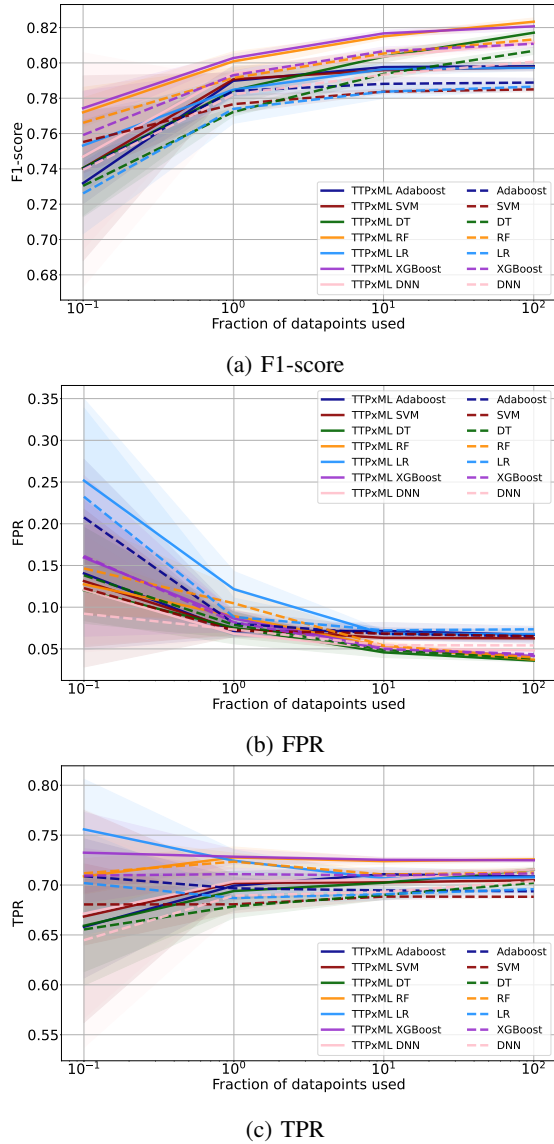


Fig. 1: Performance when varying the size of the training set. On the x-axis the percentage of randomly sampled datapoints from the training set ($\{0.1\%, 1\%, 10\%, 100\%\}$) can be found.

few datapoints are likely to be available for novel malware, whenever it is discovered. To test our hypothesis, we randomly sample from our datasets, and from each of them we generate 3 train sets with 0.1%, 1%, and 10% of the datapoints, which we consider together with the full datasets with 100% of the datapoints.

We train our models on the newly generated datasets, and plot the average of the results (obtained on the original full test sets) in Fig. 1. We also plot the 95% confidence interval for each result, which is represented by the shading around each line in the figure. As expected, the confidence intervals become smaller as the dataset size grows. Focusing on the averages, we first confirm that the TTP-based features are particularly helpful in data scarce situations, since the TTPxML models outperform the standard models to an even greater extent when lesser data is available. Then, we notice that for most

metrics and training set sizes, adding the TTPs results in an improvement of the average performance. Indeed, out of the 112 comparisons between the average results (4 dataset sizes \times 7 models \times 4 metrics) we find that the TTPxML models outperform the standard ones 96 times (i.e., 85.7% of the time). The 16 cases where the standard models do better are all statistically not significant, as the paired t-tests for these results do not return a p-value < 0.05 , with the exception of one (SVM at percentage 0.1 for TPR, where the t-test returns a p-value of 0.02). Further, in these 16 cases, the difference in performance is always negligible, with the only exception given by the performance of the logistic regression model, as judged by the FPR, which favours the standard model when training with 0.1% and 1% of the training dataset. This though is compensated by a significant improvement in performance as judged by the TPR, where we see that the model trained with TTP-based features shows an improvement of 7.66% over the standard model when training with 0.1% of the datapoints, and by 5.44% when training with 1% of the datapoints. We further observe that irrespective of the ML model chosen, the TTPxML models obtain equal or better results than the standard models while using an order of magnitude lesser data.

B. Robustness to Bin Size Variation

We now check whether the described improvements depend on the size of the chosen BoF representation. To this end, for each of the 5 balanced datasets, we vary the number of bins used to create the vectorial representation of each sample from 10 to 200 with step 10. We train each model on the BoF representations obtained with the different bin sizes, and plot the average performance over the 5 datasets in Fig. 2 (together with the performance for bin size 5, which is the size used in all our prior experiments). Again, we represent the 95% confidence interval with the shading around each average line.

Looking at Fig. 2a and 2c, we observe that the performance of all the models tends to increase as the bin size increases. In particular, we have a steeper rise from bins of size 5 to 25, and then from 30 onward the performance either stabilises or grows more slowly. However, looking at Fig. 2b we see the opposite trend: for most models we get the best (i.e., the lowest) FPR with bin size equal to 5, which then steeply increases between 5 and 25, and finally stabilises from bin size 30 onward for all models (with the exception of the decision trees for which the FPR keeps growing). The low FPR—together with the lower training and testing time for all ML models—explains why we chose bin size as 5 in the rest of our experiments.

Given the general trend followed by all the models, we now focus on the difference in performance between the TTPxML models and the standard ones. If we look at each metric, we see that (i) in terms of both F1-score and accuracy all the TTPxML models perform better than their standard counterparts for all bin sizes, and (ii) in terms of FPR and TPR all the TTPxML models perform better than their standard counterparts for all bin sizes with the exception of 5 and 4 instances, respectively. This is particularly impressive, as it highlights that out of all the 588 comparisons (21 bin sizes \times

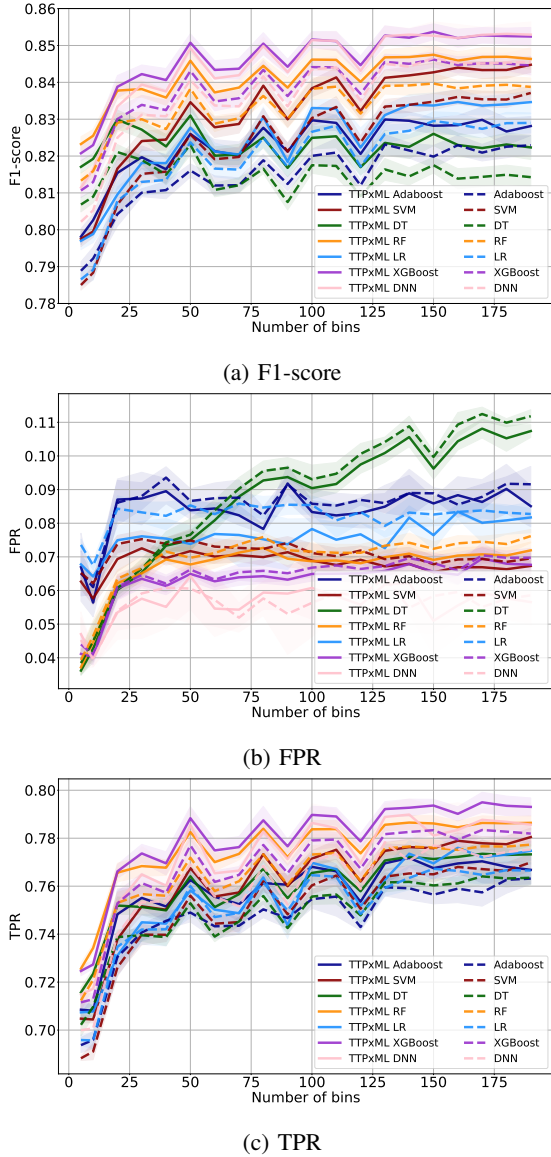


Fig. 2: Performance when varying the size of the bins to create the BoF. On the x-axis the size of the bins can be found.

7 models \times 4 metrics), the standard models manage to achieve better results than the TTPxML models only 9 times (i.e., the TTPxML models outperform them 98.4% of the time). Further, since in all of these 9 cases the difference in performance is again negligible (minimum difference equal to 8.4×10^{-5} and maximum difference equal to 0.006), we again perform a paired t-test to check the statistical significance of the results. As expected, all the p-values were > 0.5 but one (for the TPR metric with bin size 90 and model logistic regression for which the p-value is ~ 0.03), and thus all such results were not statistically significant with a single exception.

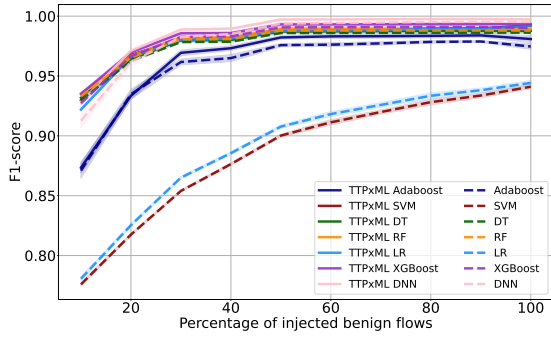
C. Robustness to Injection of Benign Flows

We consider an adversarial scenario, where the adversary attempts to evade detection by camouflaging their malicious activity using benign activities. To simulate this situation, we follow the approach proposed in [13] and inject varying

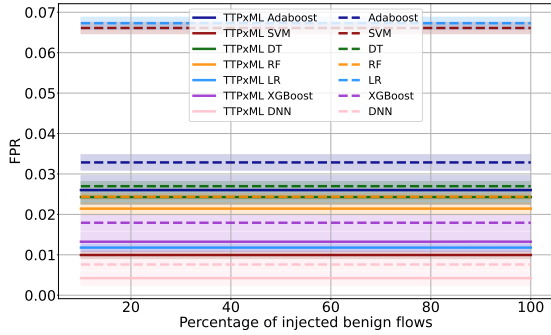
amounts of benign flows into our malicious samples. Thus, for each of the 5 balanced training datasets, we obtain a novel training set by injecting different percentages of benign flows in the malicious samples. In particular, we consider percentage rates from 10% (i.e., given a malicious sample with N flows, we add $\lfloor \frac{N}{10} \rfloor$ random benign flows) to 100%, with step size 10, thus obtaining 10 different test sets for each of the 5 initial test sets. Our choice has two advantages: (i) it simulates a realistic scenario where we do not know what percentage of benign flow injection a malicious actor might use, and (ii) it allows us to evaluate whether the model is truly learning to detect malware traffic patterns, rather than potentially overfitting on the distribution of the specific benign dataset used.

We plot the average performance (together with the 95% confidence interval) of our models when varying the levels of benign injection in Fig. 3. We immediately notice that: (i) the performance of all the models in terms of F1-score, accuracy and TPR increases as the percentage of benign flows injected in the malicious sample increases, and (ii) the FPR remains constant throughout the experiment. Regarding the former, it might seem counter-intuitive to observe that the task becomes easier as more benign flows are injected in the malicious samples. However, upon inspection, we notice that the average number of flows for malicious samples (with no benign flows injection) is 5 times higher than the average number of flows of the benign samples. Thus, injecting benign flows into the malicious samples makes the difference between malicious and benign samples even more striking. Notice that our models are able to pick up such trend due to our BoF representation, which provides a holistic view of each sample. On the contrary, in works such as [13], camouflaging has a very negative impact on the performance of the model, as there the model only has access to fixed length sequences of flows. Regarding our second observation, the stable FPR is expected since the amount of benign data in each test set is constant.

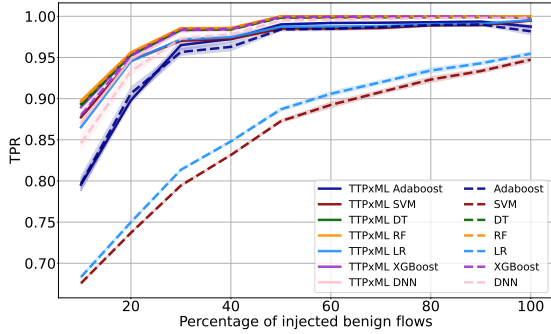
Let us now focus on the comparison between the TTPxML models and the standard ML ones. As is evident from Fig. 3, the TTP-based features are also helpful in this scenario. Indeed, we find that out of the 252 comparisons of the average performance (9 percentages \times 7 models \times 4 metrics) between the two types of models, the TTPxML models outperform the standard ML ones 247 times (i.e., 98% of the time). Further, in the 5 cases where it is not so, the difference in performance is negligible (minimum difference equal to 0.0001 and maximum difference equal to 0.008). We also perform a paired t-test to check the statistical significance of the results, and as expected, none of them return a p-value < 0.05 , i.e., such differences are not statistically significant. On the contrary, we can see that for some models the introduction of TTP-based features results in a significant improvement of performance. In particular, for logistic regressions, the TTPxML model's improvement over the standard model reaches a maximum of (i) 18.2% in terms of F1-score, (ii) 14.9% in terms of accuracy, (iii) 26.1% in terms of TPR, and (iv) 82.4% in terms of FPR. Similarly for SVMs, the TTPxML model's improvement over the standard model reaches a maximum of (i) 19.8% in terms of F1-score,



(a) F1-score



(b) FPR



(c) TPR

Fig. 3: Performance when injecting different percentages of benign flows in the samples at testing time. On the x-axis the percentage of injected benign flow can be found.

(ii) 15.9% in terms of accuracy, (iii) 28.4% in terms of TPR and (iv) 84.9% in terms of FPR.

D. Overall Comparison

Overall, TTPxML models consistently perform better than standard ML models: when averaging over the 5 datasets, we compare the models a total of 952 times, and the TTPxML models outperform the standard models 922 times (i.e., 96.8% of the time). Of the 30 cases in which TTPxML has worse performance, 29 of those cases are not statistically significant.

Focusing on F1-score, which is the standard metric used for comparison in the field, the TTPxML models outperform the standard ML models 235 times out of a total of 238 instances, i.e., our models have a higher F1-score 98.7% of the time. This consistent trend holds even in the best-case scenario for the standard models—100% datapoints, no injection, and

minimum bin size—albeit with smaller gains. Overall, we find that regardless of the amount of data, the size of the BoF representation, adversarial conditions, type of ML model, or metric chosen; the TTPxML models consistently outperform the standard models.

VI. DISCUSSION

Coverage of Malware. Our results show that the key factor responsible for the overall increase in performance of TTPxML models is the improvement in TPR, i.e., using the TTP-based features leads to more malicious samples being accurately identified as malicious than before. This follows from the fact that TTPs represent high-level adversarial behaviours, and thus by incorporating this knowledge in ML models, their ability to recognise such behaviour increases commensurately.

High Performance with Less Data. We also find that our approach is particularly useful in improving the performance of a model under data-scarce regimens. In particular, TTPxML models yield F1-scores greater or equal than those of standard models, even when trained with a tenth of the data used by the standard models. This has significant practical implications for cyber defence, given that malware is valued and feared as a function of its novelty—the less we know about it, or the less we have encountered it, the higher the risk it potentially poses (e.g., zero-day attacks).

Minimal Additional Data Cost. One of the most significant advantages of our approach is the fact that all this performance improvement comes at near-zero additional data cost. Most ML-based systems are data greedy and require increasing amounts of data to improve performance. To the contrary, our TTP-based features are created from existing source data. The only additional data required is the knowledge contained within malware ontologies such as the ATT&CK framework, which are easily available, limited in size, and easy to store—none of which can be said about other additional source data.

VII. CONCLUSIONS

In this paper, we propose a methodology for exploiting the domain knowledge of known adversarial behaviour, in the form of TTPs from the MITRE ATT&CK framework, for the purposes of improving ML-based malware detection on the network. Our approach automatically constructs TTP-based features from network traffic and provides this novel network sample representation as input to various ML models. We demonstrate the utility of our approach by evaluating our TTPxML models against standard ML models over 5 challenging datasets comprising of over 1.5 million samples and 37 million flows. Our results conclusively show that exploiting TTPs in this manner leads to better performance—indeed, models trained with the TTP-based features outperform the standard models across every metric, 96.8% of the time. Furthermore, TTPxML models: (i) have significantly better performance when less data is available, (ii) have similar or better performance than standard models while using a tenth of the data, and (iii) are robust in the face of adversarial attack.

REFERENCES

- [1] R. Malaiya *et al.*, “An empirical evaluation of deep learning for network anomaly detection,” *IEEE Access*, 2019.
- [2] D. Gibert *et al.*, “The rise of machine learning for detection and classification of malware,” *J. of Netw. and Comp. Apps.*, 2020.
- [3] K. Shaikat *et al.*, “A survey on machine learning techniques for cyber security in the last decade,” *IEEE Access*, 2020.
- [4] R. Bapat *et al.*, “Identifying malicious botnet traffic using logistic regression,” in *SIEDS*, 2018.
- [5] F. Khan, J. Ahamed, S. Kadry, and L. Ramasamy, “Detecting malicious urls using binary classification through adaboost algorithm,” *J. of Elec. & Comp. Eng.*, 2020.
- [6] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, “A mobile malware detection method using behavior features in network traffic,” *J. Netw. Comp. Appl.*, 2019.
- [7] D. Bekerman *et al.*, “Unknown malware detection using network traffic classification,” in *IEEE CNS*, 2015.
- [8] K. Bartos, M. Sofka, and V. Franc, “Optimized invariant representation of network traffic for detecting unseen malware variants,” in *USENIX*, 2016.
- [9] H. Huang, L. Qian, and Y. Wang, “A svm-based technique to detect phishing urls,” *Inf. Tech. Journal*, 2012.
- [10] B. Tellenbach *et al.*, “Accurate network anomaly classification with generalized entropy metrics,” *C. Net.*, 2011.
- [11] K. Soska *et al.*, “Automatically detecting vulnerable websites before they turn malicious,” in *USENIX*, 2014.
- [12] L. Bilge, D. Balzarotti *et al.*, “Disclosure: detecting botnet command and control servers through large-scale netflow analysis,” in *ACSAC*, 2012.
- [13] M. Piskozub *et al.*, “Malphase: Fine-grained malware detection using network flow data,” in *AsiaCSS*, 2021.
- [14] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, “A two-layer deep learning method for android malware detection using network traffic,” *IEEE Access*, 2020.
- [15] E. H. Do and V. N. Gadepally, “Classifying anomalies for network security,” in *IEEE ICASSP*, 2020.
- [16] C. Cordero, S. Hauke, M. Mühlhäuser, and M. Fischer, “Analyzing flow based anomaly intrusion detection using replicator neural networks,” in *PST*, 2016.
- [17] S. S. Dhaliwal, A.-A. Nahid, and R. Abbas, “Effective intrusion detection system using xgboost,” *Inform.*, 2018.
- [18] S. Bhattacharya, P. Maddikunta *et al.*, “A novel pca-firefly based xgboost classification model for intrusion detection in networks using gpu,” *Elect.*, 2020.
- [19] “ATT&CK.” [Online]. Available: <https://attack.mitre.org>
- [20] V. Chierzi *et al.*, “Evolution of IoT linux malware: A mitre att&ck ttp based approach,” in *eCrime*, 2021.
- [21] R. Al-Shaer *et al.*, “Learning the Associations of MITRE ATT&CK Adversarial Techniques,” in *CNS*, 2020.
- [22] K. Oosthoek and C. Doerr, “SoK: ATT&CK techniques and trends in windows malware,” in *SecureComm*, 2019.
- [23] J. Fairbanks *et al.*, “Identifying att&ck tactics in android malware control flow graph through graph representation learning and interpretability,” in *IEEE Big Data*, 2021.
- [24] O. Mendsaikhan, H. Hasegawa, Y. Yamaguchi, and H. Shimada, “Automatic Mapping of Vulnerability Information to Adversary Techniques,” in *SecureWare*, 2020.
- [25] V. Legoy *et al.*, “Automated retrieval of ATT&CK tactics and techniques for cyber threat reports,” *arXiv*, 2020.
- [26] G. Husari, E. Shaer *et al.*, “TTPDrill: Automatic and Accurate Extraction of Threat Actions From Unstructured Text of CTI Sources,” in *ACSAC*, 2017.
- [27] A. Kuppa, S. Grzonkowski, M. Asghar, and N. Le-Khac, “Finding rats in cats: Detecting stealthy attacks using group anomaly detection,” in *TrustCom*, 2019.
- [28] W. Hassan *et al.*, “Tactical provenance analysis for endpoint detection and response systems,” in *IEEE SP*, 2020.
- [29] Y. Sharma *et al.*, “RADAR: A TTP-based Extensible, Explainable, and Effective System for Network Traffic Analysis and Malware Detection,” *arXiv*, 2022.
- [30] Y. Huang, C. Lin, Y. Guo, K. Lo, Y. Sun, and M. Chen, “Open source intelligence for malicious behavior discovery and interpretation,” *IEEE Dep. and Sec. Comp.*, 2021.
- [31] V. Paxson, “Bro: a system for detecting network intruders in real-time,” *Comput. Netw.*, 1998.
- [32] O. A. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” *IEEE Access*, 2020.
- [33] M. Kamar *et al.*, “A survey on mobile malware detection methods using machine learning,” in *IEEE CCWC*, 2022.
- [34] C. M. Inacio and B. Trammell, “YAF: Yet Another Flowmeter,” in *LISA*, 2010.
- [35] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Pearson Prentice Hall, 2008.
- [36] D. Divakaran *et al.*, “Slic: Self-learning intelligent classifier for network traffic,” *Comput. Netw.*, 2015.
- [37] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and Y. Xiang, “Internet traffic classification by aggregating correlated naive bayes predictions,” *IEEE Inf. For. and Sec.*, 2013.
- [38] H. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware ML Models,” *arXiv*, 2018.
- [39] G. Severi, T. Leek, and B. Dolan-Gavitt, “Malrec: Compact Full-Trace Malware Recording for Retrospective Deep Analysis,” in *DIMVA*, 2018.
- [40] “MalShare.” [Online]. Available: <https://malshare.com>
- [41] “VirusShare.” [Online]. Available: <https://virusshare.com>
- [42] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” in *EuroCOLT*, 1995.
- [43] J. Quinlan, “Induction of decision trees,” *M.Learn.*, 1986.
- [44] D. R. Cox, “The regression analysis of binary sequences,” *Journal of the Royal Statistical Society*, 1958.
- [45] W. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of mathematical biophysics*, 1943.
- [46] T. Ho, “The random subspace method for constructing decision forests,” *IEEE Pattern Anal. Mach. Intell.*, 1998.
- [47] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. learn.*, 1995.