Learning Explainable Representations of Malware Behavior

Paul Prasse (✉)[1], Jan Brabec[2], Jan Kohout[2],
Martin Kopp[2], Lukas Bajer[2], and Tobias Scheffer[1]

[1] University of Potsdam, Department of Computer Science, Germany
{prasse,scheffer}@uni-potsdam.de
[2] Cisco Systems, Cognitive Intelligence, Prague, Czech Republic
{janbrabe,jkohout,markopp,lubajer}@cisco.com

Abstract. We address the problems of identifying malware in network telemetry logs and providing *indicators of compromise*—comprehensible explanations of behavioral patterns that identify the threat. In our system, an array of specialized detectors abstracts network-flow data into comprehensible *network events* in a first step. We develop a neural network that processes this sequence of events and identifies specific threats, malware families and broad categories of malware. We then use the *integrated-gradients* method to highlight events that jointly constitute the characteristic behavioral pattern of the threat. We compare network architectures based on CNNs, LSTMs, and transformers, and explore the efficacy of unsupervised pre-training experimentally on large-scale telemetry data. We demonstrate how this system detects njRAT and other malware based on behavioral patterns.

Keywords: neural networks · malware detection · sequence models · unsupervised pre-training

1 Introduction

Toady's malware can exhibit different kinds of malicious behaviour. Malware collects personal and financial data, can encrypt users' files for ransom, is used to commit click-fraud, or promotes financial scams by intrusive advertising. Client-based antivirus tools employ signature-based analysis, static analysis of portable-executable files, emulation, and dynamic, behavior-based analysis to detect malware [34]. Systems that analyze network telemetry data complement antivirus software and are widely used in corporate networks. They allow organizations to enforce acceptable-use and security policies throughout the network and minimize management overhead. Telemetry analysis makes it possible to encapsulate malware detection into network devices or cloud services [17, 6].

Research on applying machine learning to malware detection is abundant. However, the principal obstacle that impedes the deployment of machine-learning solutions in practice is that computer-security analysts need to be able to validate and confirm—or overturn—decisions to block software as malware. However, machine-learning models usually work as black boxes and do not provide a

decision rationale that analysts can understand and verify. In computer security, *indicators of compromise* refer to specific, observable evidence that indicates, with high confidence, malicious behavior. Security analysts consider indicators of compromise to be grounds for the classification of software as malware. For instance, indicators of compromise that identify software as variants of the *WannaCry* malware family include the presence of the WannaCry ransom note in the executable file and communication patterns to specific URLs that are used exclusively by a kill-switch mechanism of the virus [3].

In recent years, machine-learning models have been developed that emphasize *explainability* of the decisions and underlying representations. For instance, *Shapley values* [22], and the *DeepLift* [32] and *integrated gradients* methods [33] quantify the contribution of input attributes to the model decision. However, in order to be part of a comprehensible explanation of why software is in fact malicious, the importance weights would have to refer to events that analysts can relate to specific behavior of malicious software.

In this paper, we first discuss a framework of classifiers that detect a wide range of intuitively meaningful network events. We then develop neural networks that detect malware based on behavioral patterns composed of these behaviors. We compare network architectures based on CNNs, LSTMs, and transformers. In order to address the relative scarcity of labeled data, we investigate whether initializing the models by unsupervised pre-training improves their performance. We review how the model detects the njRAT and other malware families based on behavioral indicators of compromise.

2 Related Work

Prior work on the analysis of *HTTP logs* [25] has addressed the problems of identifying command-and-control servers [24], unsupervised detection of malware [19, 7], and supervised detection of malware using domain blacklists as labels [13, 6, 8]. HTTP log files contain the full URL string, from which a wide array of informative features can be extracted [6].

A body of recent work has aimed at detecting Android malware by network-traffic analysis. Arora *et al.* [5] use the average packet size, average flow duration, and a small set of other features to identify 48 malicious Android apps. Lashkari *et al.* [20] collect 1,500 benign and 400 malicious Android apps, extract flow duration and volume feature, and apply several machine-learning algorithms from the Weka library. They observe high accuracy values on the level of individual flows. Demontie *et al.* [10] model different types of attacks against such detection mechanisms and devise a feature-learning paradigm that mitigates these attacks. Malik and Kaushal [23] aggregate the VirusTotal ranking of an app with a crowd-sourced domain-reputation service (Web of Trust) and the app's resource permission to arrive at a ranking.

Prior work on *HTTPS logs* has aimed at identifying the application layer protocol [37, 9, 12]. In order to cluster web servers that host similar applications, Kohout *et al.* [18] developed features that are derived from a histogram

of observable time intervals and data volumes of connections. Using this feature representation, Lokoč *et al.* [21] introduced an approximate k-NN classifier that identifies servers which are contacted by malware.

Graph-based classification methods [4] have been explored, but they can only be applied by an agent that is able to perceive a significant portion of the global network graph—which raises substantial logistic and privacy challenges. By contrast, this paper studies an approach that relies only on the agent's ability to observe the traffic of a single organization.

Prior work on neural networks for network-flow analysis [26] has worked with labels for client computers (infected and not infected)—which leads to a multi-instance learning problem. CNNs have also been applied to analyzing URLs which are observable as long as clients use the HTTP instead of the encrypted HTTPS protocol [30]. Malware detection from HTTPS traffic has been studied using a combination of word2vec embeddings of domain names and long short term memory networks (LSTMs) [27] as well as convolutional neural networks [28]. Since the network-flow data only logs communication events between clients and hosts, these models act as black boxes that do not provide security analysts any verifiable decision rationale. Since we collected data containing only specific network events without the information of the used domain names, we are not able to apply these models to our data.

Recent findings suggest that the greater robustness of convolutional neural networks (CNNs) may outweight the ability of LSTMs to account for long-term dependencies [14]. This motivates us to explore convolutional architectures. Transformer networks [36] are encoder-decoder architectures using multi-head self-attention layers and positional encodings widely used for NLP tasks. GPT-2 [29] and BERT [11] show that transformers pre-trained on a large corpus learn representations that can be fine-tuned for classification problems.

3 Problem Setting and Operating Environment

This section first describes the operating environment and the first stage of the *Cisco Cognitive Intelligence* system that abstracts network traffic into *network events*. Section 3.2 proceeds to define the threat taxonomy and to lay out the problem setting. Section 3.3 describes the data set that we collect for the experiments described in this paper.

3.1 Network Events

The *Cisco Cognitive Intelligence (CI)* [35] intrusion detection system monitors the network traffic of the customer organization for which it is deployed. Initially, the traffic is captured in the form of web proxy logs that enumerate which users connect to which servers on the internet, and include timestamps and the data volume sent and received. The CI engine then abstracts log entries into a set of *network events*—high-level behavioral indicators that can be interpreted by security analysts. Individual network events are not generally suspicious by

themselves, but specific *patterns of network events* can constitute *indicators of compromise* that identify threats. In total, CI distinguishes hundreds of events; their detection mechanisms fall into four main categories.

- *Signature-based events* are detected by matching behavioral signatures that have been created manually by a domain expert. This includes detection based on known URL patterns or known host names.
- *Classifier-based events* are detected by special-purpose classifiers that have been trained on historical proxy logs. These classifiers included models that identify specific popular applications.
- *Anomaly-based events* are detected by a multitude of statistical, volumetric, proximity-based, targeted, and domain-specific anomaly detectors. Events in this category include, for example, contacting a server which is unlikely for the given user, or communication patterns that are too regular to be caused by a human user using a web browser.
- *Contextual events* capture various network behaviors to provide additional context; for instance, file downloads, direct access of a raw IP address without specified host name, or software updates.

For purposes of the work, each interval of five minutes in which at least one network flow is detected, the *set of network events* is timestamped and logged. Events are indexed by the *users* who sent or received the traffic. No data are logged for intervals in which no event occurs. The resulting data structure for each organization is a sparse sequence of sets of network events for each user within the organization.

3.2 Identification of Threats

We use a malware taxonomy with three levels: *threat ID, malware family, and malware category*. The *threat ID* identifies a particular version of a malware product, or versions that are so similar that a security analyst cannot distinguish them. For instance, a threat ID can correspond to a particular version of the njRAT malware [1], all instances of which use the same user-agent and URL pattern for communication. The *malware family* entails all versions of a malware product—for instance, WannaCry is a malware family of which multiple versions are known to differ in their communication behavior. Finally, the *malware category* broadly characterizes the monetization scheme or harmful behavior of a wide range of malware products. For instance, *advertisement injector*, *information stealer*, and *cryptocurrency miner* are malware categories.

Labeled training and evaluation data consist of sets of network events of five-minute intervals associated with a particular user in which threats have been identified by security analysts. In order to determine threat IDs, malware families, and categories, security analysts inspect network events and any available external sources of information about contacted servers. In some cases, hash keys of the executable files are also available and can be matched against databases of known malware to determine the ground truth. Due to this involvement of qualified experts, labeled data are valuable and relatively scarce.

Table 1. Data set statistics for malware category evaluation.

Malware category	Training instances	Test instances
Potentially unwanted application	14,675	10,026
Ad injector	14,434	17,174
Malicious advertising	3,287	1,354
Malicious content distribution	2,232	9,088
Cryptocurrency miner	1,114	1,857
Scareware	198	398
Information stealer	128	131

The *problem setting* for the malware-detection model is to detect for each organization, user, and each five-minute interval in which at least one network event has occurred, which threat ID, malware family, and malware category the user has been exposed to. That is, each *instance* is a combination of an organization, a user and a five-minute time interval. Threats are presented to security analysts on the most specific level on which they can be detected. Specific threat IDs provide the most concrete actionable information for analysts. However, for unknown or unidentifiable threats, the malware family or category provides a lead which an analyst can follow up on. In addition to the threat, *indicators of compromise* in the form of the relevant network events that identify the threat have to be presented to the analysts for review.

The analysis of this paper focuses on distinguishing between different threat IDs, malware families, and categories, and offering comprehensive indicators of compromise. The equally important problem of distinguishing between malware and benign activities has, for instance, been studied by Prasse *et al.* [28]. The majority of benign network traffic is not included in our data because only time intervals in which network events occur are logged.

We will measure precision-recall curves, the multi-class accuracy, and the macro-averaged AUC to evaluate the models under investigation. The average AUC is calculated as the mean of the AUC values of the individual classes. Precision—the fraction of alarms that are not false alarms—directly measures the amount of unnecessary workload imposed on security analysts, while recall quantifies the detection rate. We also compare the models in terms of ROC cuves because these curves are invariant to class ratios.

3.3 Data Collection and Quantitative Analysis

We collected the entire network traffic of 348 companies for one day in June 2020 as training data, and for one day in July 2020 as evaluation data. The training data contain the network traffic of 1,506,105 users while the evaluation data contain the traffic of 1,402,554 unique users. In total, the data set consists of 9,776,911 training instances and 9,970,560 test instances, where each instance is a combination of an organization, a user, and a five-minute interval in which at least one network event was observed. In total, 216 distinct network events

Table 2. Data set statistics for malware family evaluation.

Malware family	Training	Test	Malware category
ArcadeYum	12,051	6,231	Potentially unwanted application
Malicious Android firmware	38	30	Information stealer
njRAT	15	37	Information stealer
WannaCry	4	7	Ransomware

Table 3. Data set statistics for threat ID evaluation.

Threat ID	Training instances	Test instances	Malware category
Threat ID 1	8,900	9,710	Ad injector
Threat ID 2	900	924	Potentially unwanted application
Threat ID 3	11,894	6,075	Potentially unwanted application
Threat ID 4	641	783	Potentially unwanted application
Threat ID 5	606	425	Ad injector
Threat ID 6	392	567	Malicious advertising
Threat ID 7	2,099	9,027	Malicious content distribution
Threat ID 8	119	54	Typosquatting
Threat ID 9	282	193	Phishing

occur at least once in training and evaluation data—most of these events occur frequently. On average, 2.69 network events are observed in each five-minute interval in the training data and 2.73 events in the test data.

Table 1 shows the seven malware categories that occur in the data at least 100 times. *Potentially unwanted applications (PUAs)* are the most frequent class of malware; these free applications are mostly installed voluntarily for some advertised functionality, but then cannot be uninstalled without expert knowledge and expose the user to intrusive advertisements or steal user data. Table 2 shows all malware families that analysts have identified in our data. Most malware families fall into the category of PUA, but analysts have been able to identify a number of high-risk viruses. Comparing Tables 1 and 2 shows that for many threats, analysts are able to determine the malware category, but not the specific malware family.

Finally, Table 3 shows those threat IDs for which at least 100 instances occur in our data. In many cases, analysts identify a specific threat which is assigned a threat ID based on the malware's behavior, without being able to ultimately determine which malware family it has been derived from.

4 Models

This section develops the neural network architectures that we will explore in the experimental part of this paper. All networks process the sequence of network events provided by the detector array. In one version of the networks, we employ

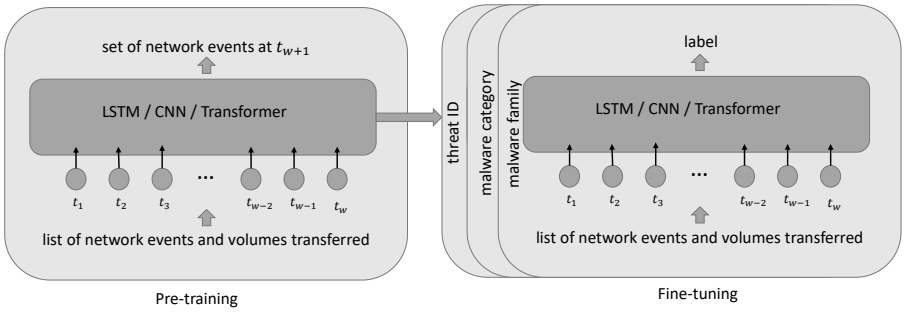


Fig. 1. Model architecture.

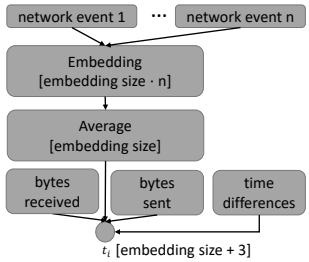
unsupervised pre-training of the models—see Figure 1. We will compare the pre-trained models to reference versions without pre-training.

4.1 Architectures

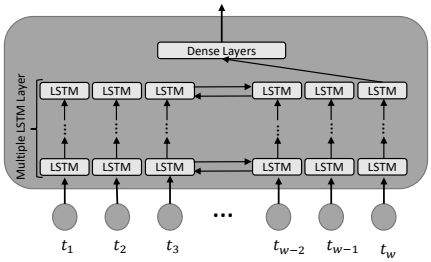
Here we develop three different model architectures: an *LSTM* model using severeral bidirectional LSTM layers [16], a *CNN* model using stacked one-dimensional CNN layers [14], and a *transformer* model that uses multiple multi-head attention blocks [36]. We also implement a random forest baseline.

The input to the different model architectures consists of a window of w five-minute intervals, each of which is represented by a set of network events, a timestamp, and the numbers of bytes sent and received. The width w of the window is a tunable hyperparameter. The set of network events for each time step are processed by an embedding layer followed by an averaging layer that computes the mean embedding for all the network events for the current five-minute interval (see Figure 2a). The mean embedding is than concatenated with the log transformed time differences between subsequent elements in the window and the log-transformed number of bytes sent and received.

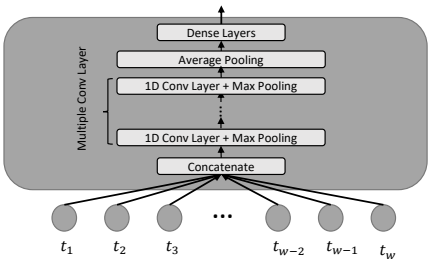
The LSTM model consists of multiple layers of bidirectional LSTM units, followed by a number of dense layers with a dropout rate of 0.1. The number of layers of each type and the number of units per layer for each of the models are hyperparameters that we will tune in Section 5; see Table 4. The output layer consists of a softmax layer with the number of units equal to the number of different classes (see Figure 2b). The CNN model starts off with a concatenation layer that combines the elements in the input window. The next layers are multiple pairs of a one-dimensional convolutional layer followed by a max-pooling



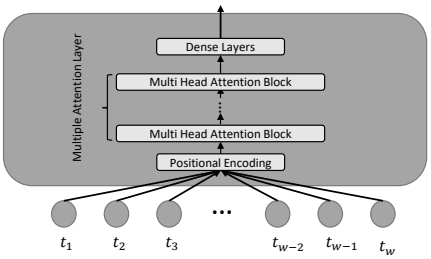
(a) Model input for a single five-minute interval.



(b) LSTM model architecture.



(c) CNN model architecture.



(d) Transformer model architecture.

Fig. 2. Model input and models.

layer. The last CNN layer is connected to an average-pooling layer and a number of dense layers on top of it. The last layer is a softmax layer with one unit per output classes (see Figure 2c).

The transformer model consists of an absolute positional encoding layer that outputs the sum of the positional encoding and the concatenated input sequence [31]. The output of the positional encoding layer is fed into multiple attention layers [36]. The output of the last multi-head attention layer is fed into a sequence of dense layers. The last layer consists of a softmax layer with one unit per output class (see Figure 2d).

The random forest (RF), which serves as natural baseline, consumes the one-hot encodings of all network events within the window and the concatenated list of all log transformed bytes sent, log-transformed bytes received, and the time differences between susequent elements within the window.

Table 4. Best hyperparameters found using grid search.

	hyperparameter	parameter range	best value
LSTM	embedding size	$\{2^6, \ldots, 2^8\}$	128
	# LSTM layers	$\{1, \cdots, 4\}$	1
	LSTM units	$\{2^3, \ldots, 2^{11}\}$	1024
	# Dense layers	$\{1, \ldots, 3\}$	2
	# Dense units	$\{2^6, \ldots, 2^{10}\}$	256
CNN	embedding size	$\{2^6, \ldots, 2^8\}$	128
	# CNN layers	$\{1, \cdots, 4\}$	3
	kernel size	$\{2^1, \ldots, 2^3\}$	4
	# filters	$\{2^2, \ldots, 2^7\}$	32
	# Dense layers	$\{1, \ldots, 3\}$	2
	# Dense units	$\{2^6, \ldots, 2^{10}\}$	256
Transformer	embedding size	$\{2^6, \ldots, 2^8\}$	128
	# attention blocks	$\{1, \cdots, 4\}$	2
	# attention heads	$\{2^2, \ldots, 2^7\}$	8
	# Dense attention units	$\{2^2, \ldots, 2^7\}$	512
	# Dense layers	$\{1, \ldots, 3\}$	2
	# Dense units	$\{2^6, \ldots, 2^{10}\}$	512
RF	# trees	$\{10, 100, 1000\}$	100
	# max depth	$\{2, 10, 100, None\}$	10

4.2 Unsupervised Pre-training

Since labeling training data requires highly-trained analysts to identify and analyze threats, labeled data are relatively scarce. While the number of labels is in the tens of thousands in our study, the number of unlabeled instances collected over two days is around 20 millions. Unsupervised pre-training offers the potential for the network to learn a low-level representation that structures the data in such a way that the subsequent supervised learning problem becomes easier to solve with limited labeled data.

To pre-train the models, we use all 9,776,911 training instances. The training objective is to predict the set of network events present at time step t_{w+1} given the sets of events of previous time steps $t_1, \ldots t_w$ (see Figure 1). This is a multi-label classification problem, since we want to predict all present network events at time step t_{w+1}. This model serves as a "language model" [29, 11] for network events that learns an internal representation which is able to predict the next network events given their context. For the pre-training step, we add a fully connected dense layer with sigmoid activation function to the models. We train these models using the binary cross entropy loss function. We will compare the *pre-trained* models to their counterparts that have been trained *from scratch* with Glorot initialization [15].

5 Experiments

This section reports on malware-detection performance of the models under investigation, and on the interpretability of the indicators of compromise. We split the data into a training part that we acquired in June 2020 and an evaluation part acquired in July 2020.

5.1 Hyperparameter Optimization

We optimize the width of the window of five-minute time intervals w used to train the models by evaluating values from 3 to 41 with a nested training-test split on the training part of the data using the threat-ID classification task. In the following experiments, we fix the number of used five-minute intervals w to 21 (see Figure 2). That is, each training and test instance is a sequence of 21 five-minute intervals; training and test sequences are split into overlapping sequences of that length. We tune the number of layers of each type, and the number of units per layer for all models using a 5-fold cross-validation on the training part of the data using the threat-ID classification task. The grid of parameters and the best hyperparameters can be found in Table 4. The optimal parameters for the random forest baseline are found using a 5-fold cross-validation on the training data of the given task.

We train all models on a single server with 40-core Intel(R) Xeon(R) CPU E5-2640 processor and 512 GB of memory. We train all neural networks using the Keras and Tensorflow libraries on a GeForce GTX TITAN X GPU using the NVidia CUDA platform. We implement the evaluation framework using the scikit-learn machine learning package. The code can be found online[3].

5.2 Malware-Classification Performance

In the following we compare the classification performance of the different models for the tasks of detecting threat IDs, malware categories, and malware families. We compare neural networks that are trained from scratch using Glorot initialization and models initialized with pre-trained weights as described in Section 4.2. We also investigate how the number of training data points per class effects the performance. To do so, we measure the accuracy acc@n and average AUC@n after the models have been trained on n instances per class. Since obtaining malware labels is time consuming and costly, this gives us an estimation of how the models behave in a few-shot learning scenario.

Table 5 shows the overall results for all described models and all the different levels of the threat taxonomy on the evaluation data. We see that the transformer outperforms CNN and LSTM most of the time, and that the pre-trained models almost always significantly outperform their counterparts that have been trained from scratch, based on a two-sided, paired t test with $p < 0.05$. Only the LSTM models are in some cases not able to benefit from pre-training. We also see that the neural network architectures outperform the random forest baseline in all settings, so we conclude that using the sequential information and sequential patterns can be exploited to classify different malware types. Using more training instances nearly always boosts the overall performance. Only for the detection of different malware families the performance in terms of the average AUC is lower when training with the full data set. We think this is caused by highly imbalanced class distribution pushing the models to favor for specific classes.

[3] https://github.com/prassepaul/Learning-Explainable-Representations-of-Malware-Behavior

From Table 5, we conclude that in almost all cases the transformer model with unsupervised pre-training is the overall best model. Because of that, the following detailed analysis is performed using only the transformer model architecture.

Additional experiments in which we determine the ROC and precision-recall curves that the transformer with pre-training achieves for individual threats, malware families, and malware categories can be found in an online appendix[4]. From these experiments, we can furthermore conclude that threat IDs that have a one-to-one relationship with a malware family are the easiest ones to identify, and that broad categories such as PUA that include a wide range of different threats are the most difficult to pin down.

Table 5. Accuracy and AUC for the detection of threat IDs, malware families, and categories, after training on some or all training data, with and without pre-training. Acc@n and AUC@n refer to the accuracy and AUC, respectively, after training on up to n instances per class. For results marked "*", the accuracy of pre-trained models is significantly better ($p < 0.05$) compared to the same model trained from scratch. Results marked "†" are significantly better ($p < 0.05$) than the next-best model.

		CNN		LSTM		Transformer		Random Forest
		Scratch	Pre-tr.	Scratch	Pre-tr.	Scratch	Pre-tr.	Scratch
threat ID	acc@10	0.394	0.437*	0.314	0.375*	0.352	**0.559***†	0.413
	acc@50	0.618	0.648*	0.567	0.478	0.612	**0.731***†	0.57
	acc@100	0.666	0.689*	0.624	0.54	0.685	**0.759***†	0.614
	acc	0.785	0.799	0.806	**0.843***†	0.769	0.776	0.809
	AUC@10	0.794	0.773	0.75	0.698	0.748	**0.848***	0.832
	AUC@50	0.889	0.893	0.874	0.807	0.893	**0.941***†	0.902
	AUC@100	0.906	0.914*	0.897	0.829	0.902	**0.948***†	0.912
	AUC	0.937	**0.952***	0.935	0.942	0.915	0.95*	0.925
malware category	acc@10	0.23	0.396*	0.196	0.312*	0.228	**0.456***†	0.338
	acc@50	0.524	0.598*	0.515	0.485	0.575	**0.652***†	0.54
	acc@100	0.618	0.669*	0.597	0.539	0.652	**0.703***†	0.606
	acc	0.77	0.785*	0.769	0.772	0.771	**0.802***†	0.73
	AUC@10	0.752	0.813*	0.73	0.728	0.747	**0.821***	0.819
	AUC@50	0.86	0.901*	0.861	0.808	0.879	**0.924***†	0.889
	AUC@100	0.881	0.91*	0.877	0.831	0.914	**0.938***†	0.907
	AUC	0.917	0.937*	0.908	0.902	0.912	**0.96***†	0.916
malware family	acc@10	0.439	**0.91***	0.01	0.894*	0.322	0.893*	0.846
	acc@50	0.839	0.939*	0.592	0.923*	0.808	**0.946***	0.923
	acc@100	0.87	0.959	0.875	0.952	0.866	**0.977***†	0.962
	acc	0.929	0.993*	0.889	**0.995***†	0.954	0.992	0.994
	AUC@10	0.886	**0.985***	0.785	0.964*	0.855	0.983*	0.106
	AUC@50	0.96	0.992*	0.832	0.982*	0.967	**0.993***	0.199
	AUC@100	0.96	0.993*	0.922	0.983	0.969	**0.995***	0.199
	AUC	0.921	**0.983***	0.923	**0.983***	0.486	0.947*	0.322

[4] https://www.uni-potsdam.de/fileadmin/projects/cs-ml/media/prasse_ecml2021_appendix.pdf

5.3 Indicators of Compromise

This section explores the interpretablility of the indicators of compromise inferred from the transformer model. We use the *integrated gradients* method to highlight the most important features for a given input sequence [33]. Integrated gradients can compute the contribution of each network event when classifying a given input sequence. We calculate the impact of all input features using

$$IG_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha, \tag{1}$$

where $_i$ denotes the i-th feature, x the input to the model, x' the baseline, and α the interpolation constant for the perturbation of the features. The term $(x_i - x_i')$ denotes the difference between original input and "baseline". Similar to the all-zeros baseline that is used for input images, we set the baseline to the instance with all zero-embeddings and original numerical features. The baseline input is needed to scale the integrated gradients. In practice we approximate this integral by the numerical approximation

$$IG_i^{approx}(x) = (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m}, \tag{2}$$

where k is the number of approximation steps.

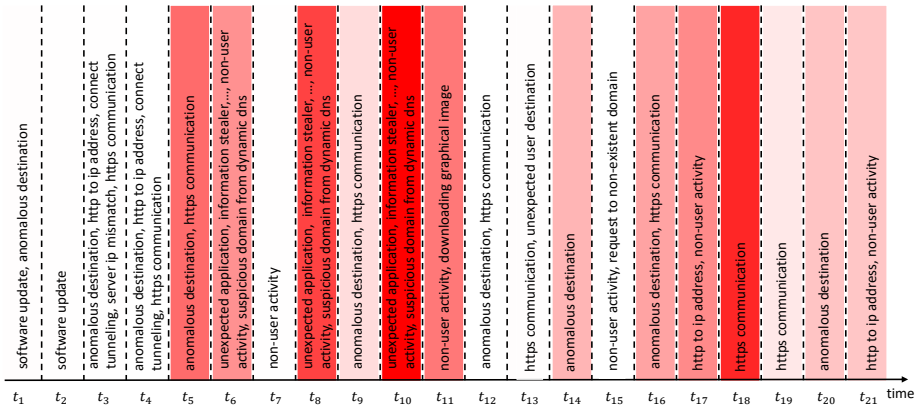
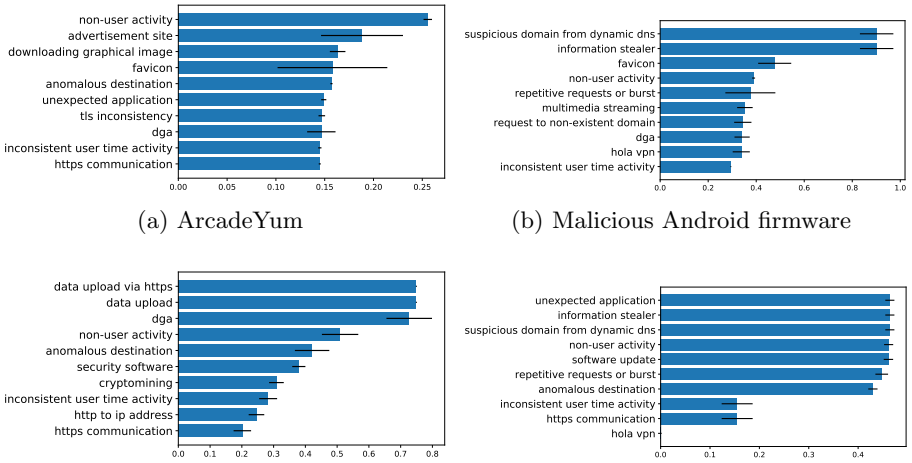


Fig. 3. Feature importance for detection of njRAT using *integrated gradients* for a single instance. The intensity of the red hue indicates the importance of network events.

Single-Instance Evaluation Using the Integrated Gradients from Equation 2, we determine which input time steps contributed to which extend to the overall classification. Figure 3 shows an example output for an instance classified as

njRAT. The njRAT malware family, also called Bladabindi, is a widespread remote access trojan (RAT). It allows attackers to steal passwords, log keystrokes, activate webcam and microphone, give access to the command line, and allows attackers to remotely execute and manipulate files and system registry.

It uses the HTTP user-agent field (this is reflected in the event *unexpected application* in Figure 3) to exfiltrate sensitive information (event *information stealer* in Figure 3) from the infected machine. The communication with C&C server uses dynamic DNS with string patterns such as *maroco.dyndns.org/israrinoy* or *man2010.no-ip.org/is-ready* and specifically crafted host names. This usage of dynamic DNS is reflected in event *suspicious domain from dynamic DNS* in Figure 3, the specific host names as event *anomalous destination*. These characteristic features of njRAT are also the most important features for the transformer. We conclude that this explanation matches known behavior of njRAT.



(a) ArcadeYum (b) Malicious Android firmware

(c) WannaCry (d) njRAT

Fig. 4. Feature importances of the top 10 features for detection of different malware families. The width of the bar is computed by using the *integrated gradients* method for each positively classified instance and averaging the obtained values for all network events. Error bars denote the standard deviation.

Feature Importance We add the feature importance values for all the instances classified as a particular malware family. Figure 4 shows the feature importance for different families. For *njRAT*, we see that the top four features captured in Figure 4d matche the behavior of njRAT described above. The *ArcadeYum* family is a typical example of the PUA/adware category. When installed, it starts to download large amounts of advertisement and present it as additional banners rendered on top of legitimate websites or as pop-up windows.

The advertisement images are downloaded on the background without users knowledge and often from hosts that may be a source of additional infections. This behaviour is again captures by the most important features in Figure 4a.

Most of the *WannaCry* samples that we were able to detect are older versions that use DGA domains as a kill switch—see [3] for details. The behavioral indicators *dga, non-user activity, anomalous destination, inconsistent user time activity* in Figure 4c are related to the regular attempt to contact these DGA domains. Some of the identified samples are actually *WannaMine* [2], a cryptomining modification of the original WannaCry malware. Their activity is captured by the *cryptomining* event as well as the *http to IP address*, which is the mechanism through which WannaMine downloads additional modules. *Malicious Android firmware*, Figure 4b, is known for gathering and exfiltrating sensitive user information and using dynamic DNS to avoid blacklists. Both behaviors are represented as the top two features. The further actions depend on the type and version of the infected device. Usually, an advertisement auction service is contacted and advertisement images or videos are being displayed (*multimedia streaming, repetitive requests, non-user activity, dga*).

6 Conclusion

We have studied the problem of identifying threats based on sequences of sets of human-comprehensible network events that are the output of a wide array of specialized detectors. We can conclude that the *transformer* architecture outperforms both the CNN and LSTM models at identifying threat IDs, malware families, and malware categories. Furthermore, unsupervised pre-training improves the transformer's performance over supervised learning from scratch. We use the *integrated gradients* method to determine the sequence of the most important network events that constitute *indicators of compromise* which can be verified by security analysts. Our detailed analysis of the njRAT malware shows that the sequence of highly important events corresponds to the known behavior of the virus. We can conclude that for the four most frequent malware families, the network events that reach the highest aggregated feature importance across all occurrences match known indicators of compromise.

References

1. Msil/bladabindi. `https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=MSIL/Bladabindi`, accessed: 2021-03-31
2. Wannamine cryptominer that uses eternalblue still active. `https://www.cybereason.com/blog/wannamine-cryptominer-eternalblue-wannacry`, accessed: 2021-03-31
3. Indicators associated with wannacry ransomware (2017 (accessed 2021-03-24)), `https://us-cert.cisa.gov/ncas/alerts/TA17-132A`
4. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T.: Graph-based malware detection using dynamic analysis. Journal of Computer Virology **7**(4), 247–258 (2011)

5. Arora, A., Garg, S., Peddoju, S.K.: Malware detection using network traffic analysis in android based mobile devices. In: International Conference on Next Generation Mobile Apps, Services and Technologies. pp. 66–71 (2014)
6. Bartoš, K., Sofka, M.: Robust representation for domain adaptation in network security. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, pp. 116–132. Springer (2015)
7. Bartoš, K., Sofka, M., Franc, V.: Optimized invariant representation of network traffic for detecting unseen malware variants. In: USENIX Security Symposium. pp. 807–822 (2016)
8. Brabec, J., Machlica, L.: Decision-forest voting scheme for classification of rare classes in network intrusion detection. In: 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 3325–3330 (2018)
9. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic classification through simple statistical fingerprinting. ACM SIGCOMM Computer Communication Review **37**(1), 5–16 (2007)
10. Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G., Roli, F.: Yes, machine learning can be more secure! a case study on android malware detection. IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2018)
11. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
12. Dusi, M., Crotti, M., Gringoli, F., Salgarelli, L.: Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. Computer Networks **53**(1), 81–97 (2009)
13. Franc, V., Sofka, M., Bartoš, K.: Learning detector of malicious network traffic from weak labels. In: Bifet, A., May, M., Zadrozny, B., Gavalda, R., Pedreschi, D., Bonchi, F., Cardoso, J., Spiliopoulou, M. (eds.) Machine Learning and Knowledge Discovery in Databases, pp. 85–99. Springer International Publishing, Cham (2015)
14. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N.: Convolutional sequence to sequence learning. In: International Conference on Machine Learning. pp. 1243–1252. PMLR (2017)
15. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 9, pp. 249–256. Chia Laguna Resort, Sardinia, Italy (13–15 May 2010)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
17. Karim, M.E., Walenstein, A., Lakhotia, A., Parida, L.: Malware phylogeny generation using permutations of code. Journal in Computer Virology **1**(1-2), 13–23 (2005)
18. Kohout, J., Pevný, T.: Automatic discovery of web servers hosting similar applications. In: Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (2015)
19. Kohout, J., Pevný, T.: Unsupervised detection of malware in persistent web traffic. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (2015)
20. Lashkari, A., Kadir, A., Gonzalez, H., Mbah, K., Ghorbani, A.: Towards a network-based framework for android malware detection and characterization. In: Proceedings International Conference on Privacy, Security, and Trust (2015)

21. Lokoč, J., Kohout, J., Čech, P., Skopal, T., Pevný, T.: k-nn classification of malware in https traffic using the metric space approach. In: Chau, M., Wang, G.A., Chen, H. (eds.) Intelligence and Security Informatics, pp. 131–145. Springer International Publishing, Cham (2016)
22. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30. pp. 4765–4774 (2017)
23. Malik, J., Kaushal, R.: CREDROID: Android malware detection by network traffic analysis. In: Proceedings of the First ACM Workshop on Privacy-Aware Mobile Computing. pp. 28–36. ACM (2016)
24. Nelms, T., Perdisci, R., Ahamad, M.: Execscent: Mining for new C&C domains in live networks with adaptive control protocol templates. In: Proceedings of the USENIX Security Symposium (2013)
25. Nguyen, T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys, Tutorials **10**(4), 56–76 (2008)
26. Pevný, T., Somol, P.: Discriminative models for multi-instance problems with tree structure. In: Proceedings of the International Workshop on Artificial Intelligence for Computer Security (2016)
27. Prasse, P., Machlica, L., Pevný, T., Havelka, J., Scheffer, T.: Malware detection by analysing network traffic with neural networks. In: Proceedings of the European Conference on Machine Learning (2017)
28. Prasse, P., Knaebel, R., Machlica, L., Pevný, T., Scheffer, T.: Joint detection of malicious domains and infected clients. Machine Learning **108**(8), 1353–1368 (2019)
29. Radford, A., Wu, J., Amodei, D., Amodei, D., Clark, J., Brundage, M., Sutskever, I.: Better language models and their implications. OpenAI Blog https://openai.com/blog/better-language-models (2019)
30. Saxe, J., Berlin, K.: eXpose: A character-level convolutional neural network with embeddings for detecting malicious urls, file paths and registry keys. arXiv preprint arXiv:1702.08568 (2017)
31. Shaw, P., Uszkoreit, J., Vaswani, A.: Self-attention with relative position representations. arXiv preprint arXiv:1803.02155 (2018)
32. Shrikumar, A., Greenside, P., Shcherbina, A., Kundaje, A.: Not just a black box: Learning important features through propagating activation differences. arXiv preprint arXiv:1605.01713 (2016)
33. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: International Conference on Machine Learning. pp. 3319–3328. PMLR (2017)
34. Swinnen, A., Mesbahi, A.: One packer to rule them all: Empirical identification, comparison and circumvention of current antivirus detection techniques. BlackHat USA (2014)
35. Valeros, V., Somol, P., Rehak, M., Grill, M.: Cognitive threat analytics: Turn your proxy into security device. Cisco Security Blog (2016 (accessed 2021-03-24)), `https://blogs.cisco.com/security/cognitive-threat-analytics-turn-your-proxy-into-security-device`
36. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017)
37. Wright, C.V., Monrose, F., Masson, G.M.: On inferring application protocol behaviors in encrypted network traffic. Journal of Machine Learning Research **7**, 2745–2769 (2006)