Malware Detection by Analysing Encrypted Network Traffic with Neural Networks

Paul Prasse[1], Lukáš Machlica[2], Tomáš Pevný[2], Jiří Havelka[2], and Tobias Scheffer[1(✉)]

[1] Department of Computer Science, University of Potsdam, Potsdam, Germany
{prasse,scheffer}@cs.uni-potsdam.de
[2] Cisco R&D, Prague, Czech Republic
{lumachli,tpevny,jhavelka}@cisco.com

Abstract. We study the problem of detecting malware on client computers based on the analysis of HTTPS traffic. Here, malware has to be detected based on the host address, timestamps, and data volume information of the computer's network traffic. We develop a scalable protocol that allows us to collect network flows of known malicious and benign applications as training data and derive a malware-detection method based on a neural embedding of domain names and a long short-term memory network that processes network flows. We study the method's ability to detect new malware in a large-scale empirical study.

1 Introduction

Malware violates users' privacy, harvests passwords, can encrypt users' files for ransom, is used to commit click-fraud, and to promote political agendas by popularizing specific content in social media [1]. Several different types of analysis are being used to detect malware.

The analysis of an organization's network traffic complements decentralized antivirus software that runs on client computers. It allows organizations to enforce a security policy consistently throughout an entire network and to minimize the management overhead. This approach makes it possible to encapsulate malware detection into network devices or cloud services. Network-traffic analysis can help to detect polymorphic malware [2] as well as new and as-yet unknown malware based on network-traffic patterns [3,4].

When the URL string of HTTP requests is not encrypted, one can extract a wide range of features from it on which the detection of malicious traffic can be based [5]. However, the analysis of the HTTP payload can easily be prevented by using the encrypted *HTTPS* protocol. Google, Facebook, LinkedIn, and many other popular sites encrypt their network traffic by default. In June 2016, an estimated 45% (and growing) fraction of all browser page loads use HTTPS [6]. In order to continue to have an impact, traffic analysis has to work with HTTPS traffic.

On the application layer, HTTPS uses the HTTP protocol, but all messages are encrypted via the Transport Layer Security (TLS) protocol or its predecessor,

M. Ceci et al. (Eds.): ECML PKDD 2017, Part II, LNAI 10535, pp. 73–88, 2017.
https://doi.org/10.1007/978-3-319-71246-8_5

the Secure Sockets Layer (SSL) protocol. An observer can see the client and host IP addresses and ports, and the timestamps and data volume of packets. Network devices aggregate TCP/IP packets exchanged between a pair of IP addresses and ports into a *network flow* for which address, timing, and data-volume information are saved to a log file. Most of the time, an observer can also see the unencrypted host domain name; we will discuss the underlying mechanisms in Sect. 3. The HTTP payload, including the HTTP header fields and the URL, are encrypted. Therefore, malware detection has to be based on properties of the host-domain names that a client contacts, and on statistical patterns in the timing and data-volumes of the sequence of network flows from and to that client. We will pursue this goal in this paper.

In order to extract features from the domain name, we explore a low-dimensional neural embedding [7] of the domain-name string. As a baseline, we will study manually-engineered domain features [5]. In order to model statistical patterns in the sequence of network flows, we will employ *long short-term memory (LSTM)* networks [8]. LSTMs are a neural model for sequential data that account for long-term dependencies in information sequences. As reference method, we will explore random forests.

The effectiveness of any machine-learning approach crucially depends on the availability of large amounts of labeled training data. However, obtaining ground-truth class labels for HTTPS traffic is a difficult problem—when the HTTP payload is encrypted, one generally cannot determine whether it originates from malware by analyzing the network traffic in isolation. We will make use of a VPN client that is able to observe the executable files that use the network interface to send TCP/IP packets. We deploy this VPN client to a large number of client computers. This allows us to observe the associations between executable files and network flows on a large number of client computers. We use antivirus tools to determine, in retrospect, which network flows in our training and evaluation data originate from malware.

This paper is organized as follows. Section 2 discusses related work. Section 3 describes our application environment and our data collection. Section 4 describes the problem setting and Sect. 5 describes our method. Section 6 presents experiments; Sect. 7 concludes.

2 Related Work

Prior work on the analysis of *HTTP logs* [9] has addressed the problems of identifying command-and-control servers [10], unsupervised detection of malware [11], and supervised detection of malware using domain blacklists as labels [5,12]. HTTP log files contain the full URL string, from which a wide array of informative features can be extracted [5]. In addition, each HTTP log file entry corresponds to a single HTTP request which also makes the timing and data volume information more informative than in the case of HTTPS, where the networking equipment cannot identify the boundaries between requests that pass through the same TLS/SSL tunnel.

Prior work on *HTTPS logs* has aimed at identifying the application layer protocol [13–15], identifying applications that are hosted by web servers [16], and identifying servers that are contacted by malware [17]. Some methods process the complete sequence of TCP packets which is not usually recorded by available network devices. Lokoč et al. [17] use similar features to the ones that we use—that can easily be recorded for HTTPS traffic—and a similar method for generating labeled data based on a multitude of antivirus tools. However, they focus on the problem of identifying servers that are contacted by malware.

Prior work on neural networks for network-flow analysis [18] has worked with labels for client computers (infected and not infected)—which leads to a multi-instance learning problem. By contrast, our operating environment allows us to observe the association between flows and executable files. Prasse et al. [19] have explored HTTPS analysis using random forests in a preliminary report. We have presented some of the results of this paper to a computer-security audience in a prior workshop paper [20].

LSTM networks [8] are widely used for translation, speech recognition and other natural-language processing tasks. Their ability to process sequential input and to account for long-term dependencies makes them appealing for the analysis of network flows. In computer security, their use has previously been explored for intrusion detection [21].

3 Operating Environment

This section describes our application and data-collection environment. In order to protect all computers of an organization, a Cloud Web Security (CWS) service provides an interface between the organization's private network and the internet. Client computers establish a VPN connection to the CWS service. The service enforces the organization's security policy; it can block HTTP and HTTPS requests based on the host domain and on the organization's acceptable-use policy. The CWS service can issue warnings when it has detected malware on a client. Since administrators have to process the malware warnings, the proportion of false alarms among all issued warnings has to be small.

The CWS service aggregates all TCP/IP packets between a single client computer, client port, host IP address, and host port that result from a single HTTP request or from the TLS/SSL tunnel of an HTTPS request into a network flow. This information is available for network devices that support the IPFIX [22] and NetFlow [23] formats. For each network flow, a line is written into the log file that includes data volume, timestamp, domain name, and duration information. For unencrypted HTTP traffic, this line also contains the full URL string. For HTTPS traffic, it includes the domain name—if that name can be observed via one of the following mechanisms.

Clients that use the Server Name Indication protocol extension (SNI) publish the unencrypted host-domain name when they establish the connection. SNI is widely used because it is necessary to verify certificates of servers that host multiple domains, as most web servers do. When the network uses a transparent

DNS proxy [24], this server caches DNS request-response pairs and can map IP addresses to previously resolved domain names. To further improve the availability of host-domain names the CWS could—but does not currently—employ passive DNS replication [25] and build a look-up table of observed DNS request-response pairs. The resulting sequence of log-file lines serves as input to the malware-detection model.

3.1 Data Collection

Since the proportion of HTTP versus HTTPS traffic declines continuously, we want to study malware detection in encrypted traffic. In our data collection, we therefore discard the URLs of HTTP requests from the log files, which leaves us only with information that would still be visible if all traffic were encrypted.

In order to obtain training and evaluation data, we have to label network traffic by whether it originated from malicious or benign applications. The CWS service can be configured to inspect HTTPS connections, provided that the service's root certificate has been installed as a trusted certificate on all client computers. This allows the CWS service to act as a man-in-the-middle between client and host, where it can decrypt, inspect, and re-encrypt HTTPS requests. This inspection can only be carried out for a small proportion of clients; we rely on it to collect training data. However, the deployed malware-detection model only uses observable features of HTTPS traffic and therefore does not require this interception. The VPN client that runs on the client computer has access to the process table and the network interface. The VPN client identifies applications by means of a SHA hash code of their executable file, and can be configured to communicate the association between HTTP/HTTPS traffic and applications to the CWS server. This allows us to augment all network flows of clients that operate with this configuration with a hash key of the application that has sent and received the network traffic. It also allows us to observe host-domain names even when SNI and transparent DNS proxy servers are not used. We configure all VPN clients of a number of participating organizations accordingly.

We label the traffic in retrospect, after the malware status of the files has been established by signature-based antivirus tools. Virustotal.com is a web service that allows users to upload executable files or hash keys of executable files. The files are run through 60 antivirus solutions, and the results of this analysis are returned. We upload the hash keys of all executable files that have generated HTTP/HTTPS traffic to Virustotal; we label files as benign when the hash is known—that is, when the file has been run through the scanning tools—and none of the 60 scanning tools recognize the file as malicious. When three or more tools recognize the file as malicious, we label it as malicious. When only one or two virus scanners recognize the file as a virus, we consider the malware status of the file *uncertain*; we do not use uncertain files for training and skip them during evaluation. In order to limit its memory use, Virustotal removes hashes of benign files from its database after some time. Therefore, we label all files whose hashes are unknown to Virustotal as benign. We then label all traffic that has been generated by malicious executables as malicious, and all traffic of benign files as benign.

We collect three different data sets; we will refer to them as *current data*, *future data*, and an independent set of *training data for domain-name features*, based on the roles which these data sets will play in our experiments. The *current data* contains the complete HTTP and HTTPS traffic of 171 small to large computer networks that use the Cisco AnyConnect Secure Mobility Solution for a period of 5 days in July 2016. This data set contains 44,348,879 flows of 133,437 distinct clients. The *future data* contains the complete HTTP and HTTPS traffic of 169 small to large different computer networks that use the Cisco AnyConnect Secure Mobility Solution for a period of 8 days in September 2016. The data set contains 149,005,149 flows of 177,738 distinct clients. The *training data for domain-name features* contains the HTTPS traffic of 21 small to large computer networks, collected over 14 days between February and April 2016. All data sets have been anonymized—information about organizations and user names have been replaced by random keys—and full URLs of HTTP requests have been removed.

3.2 Quantitative Analysis of the Data

We query the malware status of all observed hash keys in the current and future data sets from Virustotal after the collection of the respective data set has been completed, and reiterate the queries to Virustotal in February 2017. We study the stability of this labeling procedure. Table 1 shows that over this period of five months for the *future data* and seven months for the *current data*, 10% of all previously uncertain files (classified as malware by one or two antivirus tools) change their status to benign, while 3.5% change their status to malicious. From all initially unknown files, 2.3% become known as benign while 0.2% become known as malicious. From the files initially labeled as benign, 1.5% become uncertain and only 0.07% become known as malicious. Likewise, only 0.16% of all initially malicious files change their label to benign. We use the labels obtained in February 2017 in the following. We can conclude that once a file has been labeled as benign or malicious by at least 3 antivirus tools, it is very unlikely that this classification will later be reversed.

The current data contains 20,130 unique hashes, 350,220 malicious and 43,150,605 benign flows; the future data 27,263 unique hashes, 955,037 malicious and 142,592,850 benign flows. Table 2 enumerates the types and frequency of different types of malware families, according to the public classification of antivirus vendors. A large proportion of malware files is classified as *potentially unwanted applications (PUAs)*. PUAs are usually bundled to freeware or shareware applications; depending on how a distributor chooses to employ these programs, they can change the browser settings and starting page, install tools bars and browser extensions, display advertisements, and can also reveal passwords to the attacker. They typically cannot be uninstalled without a virus removal tool or detailed technical background knowledge.

Table 1. Virustotal malware status confusion matrix

Status July/September 2016	Status February 2017			
	uncertain	unknown	benign	malicious
uncertain	6,507	8	746	260
unknown	61	17,371	411	38
benign	949	38	58,899	40
malicious	150	0	8	4,781

Table 2. Malware families

Malware family	Type	Variations
dealply	adware	506
softcnapp	adware, PUA	119
crossrider	adware, PUA	98
elex	adware, PUA	86
opencandy	adware, PUA	57
conduit	adware, spyware, PUA	56
browsefox	PUA	52
speedingupmypc	PUA	29
kraddare	adware, PUA	28
installcore	adware, PUA	27
mobogenie	PUA	26
pullupdate	PUA	25
iobit downloader	adware, PUA	24
asparnet	trojan, adware	24

4 Client Malware Detection Problem

We will now establish the malware-detection problem setting. Our goal is to flag client computers that are hosting malware. Client computers are identified by a (local) IP address and a VPN user name.

For each interval of 24 h, we count every client computer that establishes at least one network connection as a separate *classification instance*; a client that is active on multiple days constitutes multiple classification instances. Each classification instance has the shape of a *sequence* $x_1, \ldots, x_T$ of *network flows* from and to a particular client on a particular day. This sequence generally blends the network flows of multiple applications that run on the client computer. When at least one malicious application generates any network traffic throughout a day, we label that instance as positive; when at least one benign application but no malicious application generates traffic, the instance is negative.

For each classification instance, malware detection model f has to decide whether to raise an alarm. The model processes a *sequence* $x_1, \ldots, x_T$ of

network flows from and to a particular client in an online fashion and raises an alarm if the malware-detection score $f(x_1, \ldots, x_t)$ exceeds some threshold τ at any time t during the day; in this case, the instance counts as a (true or false) positive. In each 24-h interval, model f with threshold τ will detect some proportion of malware-infected clients, and may raise some false alarms. The trade-off between the number of detections and false alarms can be controlled by adjusting threshold τ. Increasing the threshold decreases the number of detections as well as the number of false alarms.

We will measure precision-recall curves because they are most directly linked to the merit of a malware detection method from an application point of view. However, since precision recall curves are not invariant in the class ratio, we will additionally use ROC curves to compare the performance of classifiers on data sets with varying class ratios.

We will measure the following performance indicators.

1. *Recall* $R = \frac{n_{TP}}{n_{TP}+n_{FN}}$ is the proportion of malicious instances that have been detected, relative to all (detected and undetected) malicious instances.
2. *Precision* $P = \frac{n_{TP}}{n_{TP}+n_{FP}}$ is the proportion of malicious instances that have been detected, relative to all (malicious and benign) instances that have been flagged. Note that the absolute number of false alarms equals $(1 - P)$ times the absolute number of alarms; a high precision implies that the number of false alarms is small and the detection method is practical.
3. The recall at a specific precision, $R@x\%P$, quantifies the proportion of malicious instances that are detected when the threshold is adjusted such that at most $1 - x\%$ of all alarms are false alarms.
4. The precision-recall curve of decision function f shows the possible tradeoffs that can be achieved by varying decision threshold τ. The precision recall curve of a fixed decision function will deteriorate if the class ratio shifts further toward the negative class.
5. The false-positive rate $R_{FP} = \frac{n_{FP}}{n_{FP}+n_{TN}}$ is the proportion of benign software that is classified as malicious. Note that the absolute number of alarms equals R_{FP} times the number of benign instances. Since the number of benign instances is typically huge, a seemingly small false-positive rate does not necessarily imply that a detection method is practical.
6. The ROC curve displays the possible tradeoffs between false-positive rate and recall that can be achieved by varying decision threshold τ. It is invariant in the class ratio but it conveys no information about the proportion of alarms that are in fact false alarms. To visualize the decision function behavior for small false-positive rates, we use a logarithmic x-axis.
7. The *time to detection* measures the interval from the first network flow sent by a malicious application on a given day to its detection.

Note the difference between *precision* and the *false-positive rate*—the risk that a benign file is mistakenly classified as malware. For instance, at a false-positive rate of 10%, the expected number of false alarms equals 10% of the number of benign instances; hence, false alarms would by far outnumber actual malware detections. By contrast, at a precision of 90%, only 10% of all alarms would be false alarms.

Training data for model f consists of labeled sequences $S = \bigcup_{i=1}^{n}\{((x_{i1}, y_{i1}), \ldots, (x_{iT_i}, y_{iT_i})\}$ in which each flow x_{it} is labeled by whether it has been sent or received by a malicious ($y_{it} = +1$) or benign ($y_{it} = -1$) application.

5 Network-Flow Analysis

This section presents our method that flags malware-infected client computers based on their network traffic.

5.1 Flow Features

The detection model processes *sequences of flows* $x_1, \ldots, x_T$ sent to or received by one particular client computer. This sequence is a blend of the network traffic of multiple applications. When a computer hosts malware, benign applications will generally interfere with any patterns in the malware's communication. For each flow, a client identifier (IP address and VPN user name), host address, ports, a timestamp, inbound and outbound data volume, and a duration are observable. From each flow x_t, we extract a vector $\phi(x_t)$ that includes the log-transformed duration, log-transformed numbers of sent and received bytes, duration, and the time gap from the preceding flows.

5.2 Domain-Name Features

Each flow contains an unencrypted host IP address. For most HTTPS connection requests, the host-domain name is visible. Otherwise, the host domain name string consists of a numeric IP address. We explore several types of features that can be extracted from the host-domain name.

Engineered Features. Franc et al. [12] develop a comprehensive set of 60 features of URL strings for malware detection in unencrypted HTTP traffic—here, the entire URL is visible to third parties. Their features include the ratio of vowel changes, the maximum occurrence ratio of individual characters for the domain and subdomain, the maximum length of substrings without vowels, the presence of non-base-64 characters, the ratio of non-letter characters, and many other characteristics. We extract the vector of these *engineered* domain-name features for all domains.

Character *n*-gram Features. Character n-gram features decompose the domain string into sets of overlapping substrings; for instance, "example.com" is composed of the 3-g "exa", "xam", "amp", ..., ".co", and "com". The number of n-grams that occur in URLs grows almost exponentially in n; in our data, there are 1,583 character 2-g, 54,436 character 3-g, and 1,243,285 character 4-g. If we added all character 3-g features to the feature representation $\phi(x_t)$ of a flow, then the total number of features of an entire sequence of T flows would impose a heavy computational burden, and cause overfitting of the malware-detection model. In our experiments, we therefore explore character 2-g features.

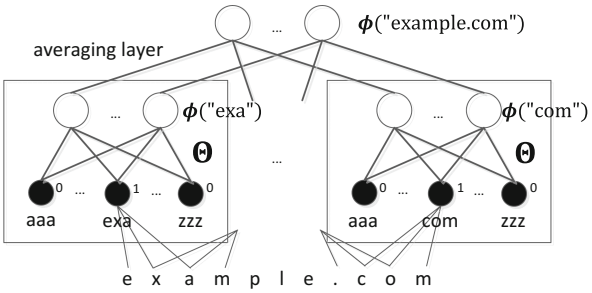


Fig. 1. Continuous bag-of-words architecture

Neural Domain-Name Features. We condense the character n-grams by means of a *neural embedding*. Neural language models [7] derive low-dimensional, continuous-state representations of words which have the property that words which tend to appear in similar contexts have similar representations. Here, "words" are the overlapping character n-grams that constitute a domain name. We apply neural embeddings with the goal of finding a representation such that substrings that tend to co-occur in URL strings have a similar representation.

We use the continuous bag-of-words architecture illustrated in Fig. 1. The input to the network consists of character n-grams that are one-hot coded as a binary vector in which each dimension represents an n-gram; Fig. 1 illustrates the case of $n = 3$. The input layer is fully connected to a hidden layer; we find that 100 hidden units give the best experimental results. The same weight matrix is applied to all input character n-grams. The activation of the hidden units is the vector-space representation of the input n-gram of characters. In order to infer the vector-space representation of an entire domain-name, an "averaging layer" averages the hidden-unit activations of all its character n-grams.

Neural language models are trained to complete an initial part of a sentence, which forces the network to find a vector-space representation that allows to "guess" a word's context from its vector-space representation. The "natural" reading order of a domain string is from the right to the left, because the domain ends with the most general part and starts with the most specific subdomain. Therefore, we use the preceding character n-gram, one-hot coded, as training target.

We train the neural language model using all domain names in our *training data for domain-name features*, and the 500,000 domains that have the highest web traffic according to alexa.com. For each URL and each position of the sliding window, we perform a back-propagation update step using the character n-gram that immediately precedes the input window as prediction target. We use the word2vec software package [26].

5.3 Client Classifier

We develop a client classifier based on long-short term memory networks (LSTMs) and, as reference method, a classifier based on random forests. LSTMs process input sequences iteratively. Each LSTM unit has a memory cell which allows it to store results of inferences; the unit may refer to this memory cell in later time steps. This allows LSTMs to account for long-term dependencies in sequential input.

Both classifiers split each client's traffic into sequences of 10 flows—for the LSTM, a fixed upper-bound on the sequence length is not necessary in theory, but it allows for more efficient training. In preliminary experiments, we have observed the performance of both LSTMs and random forests to deteriorate with longer sequences. After 10 flows $x_1, \ldots, x_{10}$ have been observed for a client, the sequence is processed by the neural network. The feature representation of each flow (duration, bytes sent and received, and the neural host domain-name representation) is processed sequentially by a layer of 32 LSTM units. After processing the input sequence, the LSTM layer passes its output to a layer of 128 cells with ReLU activation function, which is connected to two softmax cells whose activation represents the decision-function scores of the classes malicious and benign. The softmax layer is trained with 50% dropout regularization.

During training, the target label for a sequence of 10 flows is positive if at least one of the 10 flows originates from a malicious application. At application time, the overall decision-function value $f(x_1, \ldots, x_T)$ for the client's full traffic over a 24-h interval that is compared against threshold τ is the maximum activation of the positive output cell over all adjacent sequences of 10 flows.

For the random-forest classifier, the feature representation of 10 flows is stacked into a feature vector $\Phi(x_1, \ldots, x_{10}) = [\phi(x_1) \ldots \phi(x_{10})]^\top$ which serves as input to the random-forest classifier. At training time, the target label of a sequence is positive if at least one flow originates from a malicious application, and at application time the decision-function value of the random forest is maximized over all 10-flow sequences.

6 Experiments

We will first study the capability of the neural domain-name features to discriminate between benign and malicious domains. We will then explore the contribution of different types of features to malware detection, and the relative performance of LSTMs versus random forests. We will study the classifiers' ability to detect malware in current and future data, and will investigate how this performance varies across known and unknown malware families.

6.1 Classification of Host Domains

In our first experiment, we investigate the types of domain-name features with respect to their ability to distinguish between domains that are contacted by

Table 3. Domain classification and feature types

Feature type	R@70%P	R@80%P	R@90%P
Neural	**0.84**	**0.79**	**0.73**
Char 2-grams	0.83	0.76	0.62
Engineered	0.68	0.36	0.0
Neural+engineered	0.75	0.64	0.24

benign and domains that are contacted by malicious applications. In this experiment, domains serve as classification instances; all domains that are contacted more often by malicious than by benign software are positive instances, and all other domains are negative instances. In our training data for domain features, there are 860,268 negative (benign) and 1,927 positive (malicious) domains. A total of 3,490 domains are contacted by both malware and benign applications; many of them are likely used for malicious purposes (*e.g.,* "https://r8---sn-4g57km7e.googlevideo.com/", "https://s.xp1.ru4.com"), while others are standard services ("maps.google.com", "public-api.wordpress.com"). Malware frequently sends requests to legitimate services and uses URL forwarding techniques to reach the actual recipient domain. For 90,445 of the domains, only the IP address string is available.

We infer engineered domain-name features, character 2-g, and the vector-space representation of each domain string using the neural language model, as described in Sect. 5.2. We use 75% of the domains for training and 25% of the domains for testing; no domain is in both the training and the test data. We train a random forest classifier to discriminate positive from negative instances. Table 3 shows precision-recall trade-offs for the different set of feature types. We find that a parameter combination of $n = 6$ (input character 6-g), $m = 4$ (during training, the vector-space representation of 4 adjacent character 6-g is averaged) and $k = 100$ (the vector-space representation of a domain name has 100 dimensions) works best. Comparing the neural domain features to the raw character 2-g and the 60 engineered features in Table 3, we find that the neural features outperform both baselines. A combination of neural and engineered features performs worse than the neural features alone, which indicates that the engineered features inflate the feature space while not adding a substantial amount of additional information.

In order to analyze the domain-name classifier in depth, we look at domain-names that achieve the highest and lowest score from the random-forest classifier that uses the neural domain features. We find that a wide range of domains receive a decision-function value of 0 (confidently benign). These lowest-scoring domains include small and mid-size enterprises, blogs on various topics, university department homepages, games websites, a number of Google subdomains, governmental agencies; sites that can perhaps best be described as "random web sites". Table 4 shows the highest-scoring domains. They include numeric IP addresses, cloud services, subdomains of the YouTube and Facebook content

Table 4. Domain-names most confidently classified as malicious

https://52.84.0.111/
https://139.150.3.78/
https://uswj208.appspot.com/
https://ci-e2f452ea1b-50fe9b43.http.atlas.cdn.yimg.com/
https://pub47.bravenet.com
https://service6.vb-xl.net/
https://sp-autoupdate.conduit-services.com
https://external-yyz1-1.xx.fbcdn.net/
https://doc-14-28-apps-viewer.googleusercontent.com/
https://239-trouter-weu-c.drip.trouter.io

delivery networks, and domains that do not host visible content and have most likely been registered for malicious purposes. Based on these findings, we continue to study the neural domain-name features and exclude the engineered and character 2-g features from the following experiments.

6.2 Client Malware Detection

Learning Methods and Feature Types. We will first report on experiments in which we conduct 10-fold cross validation on the *current data*; we split the data into partitions with disjoint sets of users. We tune the parameters of the random forest using a grid search in an inner loop of two-fold cross validation on the training part of the data. The number of LSTM units is optimized in the first fold of the 10-fold cross validation on the training part of the data and fixed for the remaining folds.

Figure 2 compares the precision-recall curves of LSTMs and random forests using neural domain-name, flow, and combined features. We conclude that LSTMs outperform random forests for this problem, and that the combination of neural domain-name features and numeric flow features outperforms either set of features. We therefore exclude random forests from the remaining experiments.

Evolution of Malware. We will now explore whether a model that has been trained on the *current data* is able to detect malware in the *future data*. We train the LSTM on the entire *current data* and evaluate it on the entire *future data*. The number of LSTM units is left fixed. We compare the resulting model to the LSTM trained by 10-fold cross validation on the *current data*. Figure 3 compares precision-recall and ROC curves. Since the *future data* contains a smaller ratio of malicious instances—the prevalence of malware changes over time and over companies—the difference between the precision-recall curves is not necessarily due to a deterioration of the decision function. But a comparison of the ROC

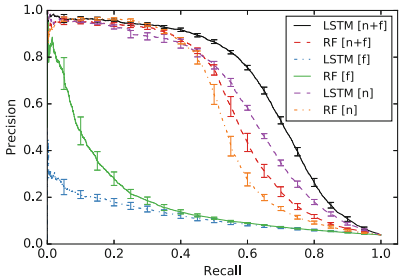
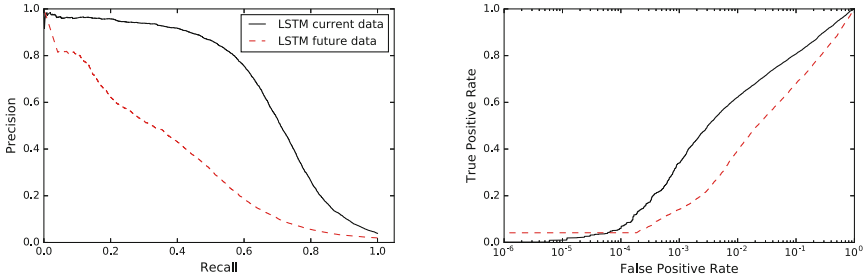


Fig. 2. Precision-recall curve for malware detection on current data. Error bars indicate standard errors.



(a) Precision-recall curves. (b) ROC curves (log-scale for FP).

Fig. 3. Comparison between performance on current and on future data.

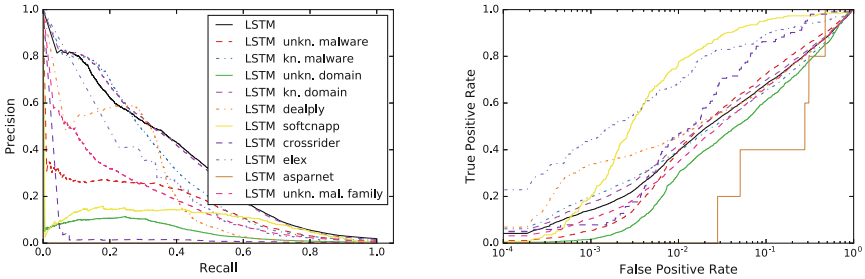
curves shows that the decision function does deteriorate to a small extent in the two months that separate the training and test data.

Malware Families. We will now study the detection performance on specific malware families, on previously unseen malware, and on malware that does not contact any previously known domain. We use an LSTM that has been trained on all *current data*. We evaluate its performance on specific subsets of malware in the *future data*. In each experiment, each user who is hosting a malicious application from a selected malware family that has sent at least one network flow within a 24-h interval counts as a positive test instance, and each user who has run at least one benign application with at least one network flow but no malicious application within a 24-h interval counts as negative classification instance. Test users who are hosting malware of different families are skipped.

Figure 4 compares precision-recall and ROC curves. Since each specific subgroup entails only few users, the class ratios are highly skewed towards the negative class and the precision-recall curves cannot be directly compared. Comparing the ROC curves, we observe that the decision function performs similarly well across the entire range of malware families, including malware that does not belong to any known family, malware that does not occur in the training

data, and malware that does not contact any domain that has been contacted by any application in the training data. The ROC curve for the malware family asparnet is fairly ragged because only 7 users in the test data are infected with.

Average Time to Detection. Table 5 compares the average intervals between the first flow sent by a malicious application and its detection for cross-validation on the *current data* and for the model that has been trained on all of the *current data* and is applied to the *future data*. Depending on the threshold, detection occurs on average approximately 90 min after malware starts communicating.



(a) Precision-recall curves. (b) ROC curves (log-scale for FP).

Fig. 4. Evaluation on user subgroups on future data.

Table 5. Average time to detection

Data set	TTC@80%P	TTC@70%P	TTC@60%P
Current	1.66 h	1.67 h	1.65 h
Future	1.40 h	1.36 h	1.38 h

7 Conclusion

We can draw a number of conclusions from our study. A neural language model can transform a domain name into a low-dimensional feature representation that provides more information about whether the site is malicious than a set of carefully engineered features of domain-name characteristics. Our experimental setting allows us to collect large volumes of malicious and benign network traffic for model training and evaluation. The VPN client records the hash key of the executable file that has generated each flow, and by using VirusTotal we are able to determine, in retrospect, which flows originate from malware.

We have developed and studied a malware-detection model based on LSTMs that uses only observable aspects of HTTPS traffic. We conclude that the LSTM-based model outperforms random forests, and that the combination of neural

domain-name features and numeric flow features outperforms either of these feature sets. This mechanism is able to identify a substantial proportion of malware in traffic that was recorded two months after the training data were recorded, including previously unseen malware. Its average time to detection is approximately 90 min after the first HTTPS request; its performance is uniform over many different malware families, including previously unknown malware. It complements signature-based and other behavior-based malware detection.

Acknowledgment. We would like to thank Virustotal.com for their kind support.

References

1. Kogan, R.: Bedep trojan malware spread by the angler exploit kit gets political. SpiderLabs Blog (2015). https://www.trustwave.com/Resources/SpiderLabs-Blog/Bedep-trojan-malware-spread-by-the-Angler-exploit-kit-gets-political/
2. Karim, M.E., Walenstein, A., Lakhotia, A., Parida, L.: Malware phylogeny generation using permutations of code. J. Comput. Virol. **1**, 13–23 (2005)
3. Gu, G., Zhang, J., Lee, W.: BotSniffer: detecting botnet command and control channels in network traffic. In: Proceedings of the Annual Network and Distributed System Security Symposium (2008)
4. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In: Proceedings of the USENIX Conference on Networked Systems Design and Implementation (2010)
5. Bartos, K., Sofka, M.: Robust representation for domain adaptation in network security. In: Bifet, A., May, M., Zadrozny, B., Gavalda, R., Pedreschi, D., Bonchi, F., Cardoso, J., Spiliopoulou, M. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9286, pp. 116–132. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23461-8_8
6. Aas, J.: Progress towards 100% HTTPS. Let's Encrypt (2016)
7. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**, 1735–1780 (1997)
9. Nguyen, T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Commun. Surv. Tutor. **10**, 56–76 (2008)
10. 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)
11. Kohout, J., Pevny, T.: Unsupervised detection of malware in persistent web traffic. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (2015)
12. Franc, V., Sofka, M., Bartos, 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.) ECML PKDD 2015. LNCS (LNAI), vol. 9286, pp. 85–99. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23461-8_6
13. Wright, C.V., Monrose, F., Masson, G.M.: On inferring application protocol behaviors in encrypted network traffic. J. Mach. Learn. Res. **7**, 2745–2769 (2006)

14. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic classification through simple statistical fingerprinting. ACM SIGCOMM Comput. Commun. Rev. **37**, 5–16 (2007)
15. Dusi, M., Crotti, M., Gringoli, F., Salgarelli, L.: Tunnel hunter: detecting application-layer tunnels with statistical fingerprinting. Comput. Netw. **53**, 81–97 (2009)
16. Kohout, J., Pevny, T.: Automatic discovery of web servers hosting similar applications. In: Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (2015)
17. 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.) PAISI 2016. LNCS, vol. 9650, pp. 131–145. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31863-9_10
18. Pevny, 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)
19. Prasse, P., Gruben, G., Machlika, L., Pevny, T., Sofka, M., Scheffer, T.: Malware detection by HTTPS traffic analysis. Technical report. urn: urn:nbn:de:kobv:517-opus4-100942 (2017)
20. Prasse, P., Machlica, L., Pevný, T., Havelka, J., Scheffer, T.: Malware detection by analysing network traffic with neural networks. In: Proceedings of the Workshop on Traffic Measurements for Cybersecurity at the IEEE Symposium on Security and Privacy (2017)
21. Staudemeyer, R., Omlin, C.: Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data. In: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (2013)
22. Claise, B., Trammell, B., Aitken, P.: Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information (2013). https://tools.ietf.org/html/rfc7011
23. Cisco Systems: Cisco IOS NetFlow (2016). http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html
24. Blum, S.B., Lueker, J.: Transparent proxy server. US Patent 6,182,141 (2001)
25. Weimer, F.: Passive DNS replication. In: Proceedings of the FIRST Conference on Computer Security Incidents, p. 98 (2005)
26. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems (2013)