Mathematisch-Naturwissenschaftliche Fakultät

Paul Prasse, Gerrit Gruben, Lukas Machlika, Tomas Pevny, Michal Sofka, Tobias Scheffer

Malware Detection by HTTPS Traffic Analysis

Preprint published at the Institutional Repository of the Potsdam University:
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-100942

Universität Potsdam

Malware Detection by HTTPS Traffic Analysis

Paul Prasse[1], Gerrit Gruben[1], Jan Kohout[2], Lukas Machlika[2],
Tomáš Pevný[2], Michal Sofka[2,3], and Tobias Scheffer[1]
[1]University of Potsdam, Germany, scheffer@cs.uni-potsdam.de
[2]Cisco R&D Prague, Czech Republic, tpevny@cisco.com
[3]now at 4Catalyzer, New York, NY, USA

***Abstract*—In order to evade detection by network-traffic analysis, a growing proportion of malware uses the encrypted HTTPS protocol. We explore the problem of detecting malware on client computers based on HTTPS traffic analysis. In this setting, malware has to be detected based on the host IP address, ports, timestamp, and data volume information of TCP/IP packets that are sent and received by all the applications on the client. 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 networks and sequence classification. We study the method's ability to detect known and new, unknown malware in a large-scale empirical study.**

I. Introduction

Malware violates users' privacy, harvests access to online-shopping and payment accounts, is used to commit click-fraud, and can encrypt users' files for ransom. Several different types of analysis are being used to detect malware, and because of the adversarial nature of the problem, robust detection requires that the problem is simultaneously attacked from different angles. Signature-based detectors employ a look-up table of software hashes, which requires individual files to first become known to be malicious through some form of analysis. Signature-based detection can be evaded by polymorphic malware that comes in an abundance of minor variations and often continues to modify its executable files after deployment [1].

Static code analysis is based on features that can be extracted from the program file. Parameters of the decision criteria are often optimized using machine-learning techniques on collections of malware and benign files [2], [3]. Static code analysis can be counteracted by code-obfuscation techniques. Here, the code that is executed is generated at execution time. Obfuscated code can be deobfuscated by partially executing it; the deobfuscated code can be subjected to static code analysis [4]. However, deobfuscation can be evaded by loading code at runtime, and by performing arbitrary complex calculations within the expressions that generate code.

In fully dynamic code analysis, the software is executed and observed for malicious behavior in a sandbox environment. Setting up a virtual operating environment, and running and observing the software is an expensive process that has to be carried out for each executable file [5]. Furthermore, the malicious behavior has to be triggered to become observable. Infected versions of standard tools may only deviate from their expected behavior in certain operating environments, or at a particular time of the day.

Malware can also be detected by analyzing network communications. TCP/IP traffic can be analyzed by network equipment without direct access to the client computer that is executing malware. This approach allows the encapsulation of malware detection into specialized network devices and and helps to protect an entire organization even if users of individual computers do not run antivirus software. Analysis of TCP/IP traffic may aim at finding specific types of malware [6], [7], or at identifying malicious servers of malware on client computers [8].

The deployment of network-traffic analysis systems has triggered evasion strategies. An analysis of the HTTP payload of the network traffic can easily be prevented by using the encrypted *HTTPS* protocol. Currently, over 40% of the most popular websites support HTTPS [9]. Under the HTTPS (HTTP over TLS/SSL) protocol, the client computer establishes a TCP/IP connection to (usually) port 443 of the host. On the application layer, HTTPS uses the standard HTTP protocol, but all messages are encrypted via the Transport Layer Security (TLS) protocol or its predecessor, the Secure Socket Layer (SSL) protocol. An observer can only see the client and host IP addresses and ports, and the timestamps and data volume of all packets. The HTTP payload, including the header fields and URL, are encrypted.

In this paper, we will develop a machine-learning method that detects malware on client computers based on the observable information of HTTPS communication. The effectiveness of machine-learning approaches 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 develop an approach to collecting training data based on a VPN client that is able to observe the associations between executable files and TCP/IP packets on a large number of client computers. One of the few observable features of HTTPS traffic is the host IP address and, if a DNS entry exists for that address, the domain name. In order to extract features from the domain name, we explore *neural language models* [10] which use neural networks to derive a low-dimensional, continuous-state representations of a text. As a baseline, we also study manually-engineered domain features [11].

The rest of this paper is organized as follows. Section II discusses related work. Section III discusses the network

architecture that constitutes the application environment for the malware detector and its operating modes. Section IV derives our method for malware detection based on HTTPS-traffic analysis. We present our empirical study in Section V. Section VI concludes.

II. Related Work

Prior work on the analysis of HTTP logs has addressed the problems of identifying command-and-control servers [12], unsupervised detection of malware [13], and supervised detection of malware using domain blacklists as labels [8], [11].

Besides the timing and volume information, HTTP log files contain the full URL string, from which a wide array of informative features can be extracted [11]. 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 and log file entries may aggregate widely varying numbers of requests.

Prior work on the analysis of HTTPS logs aims at identifying the corresponding application layer protocol [14], [15], [16], identifying applications that are hosted by web servers [17], and identifying servers that are contacted by malware [18]. Some of these methods process the complete sequence of TCP packets which is not usually recorded by available network devices. Lokoč et al. [18] use similar features to the ones that we use—features that can easily recorded for HTTPS traffic—and a similar method for generating labeled data based on a multitude of antivirus tools. However they focus on a different problem: they aim at identifying servers that are contacted by malware.

III. Threat-Analysis Architecture

This section describes the network architecture that constitutes the application environment for the malware detector—it is visualized in Figure 1. We distinguish between the regular operating environment and the environment in which we record training data.

A. Operating Environment

The network configuration consists of *client computers*, *web hosts*, a *firewall*, and a *threat analysis server*. All TCP/IP traffic from the client computer is routed through the firewall that blocks, transmits, or reroutes TCP/IP traffic that is addressed to certain IP address ranges and ports according to its configured security policy; it receives incoming traffic and forwards it via the TLS/SSL to the client computer. The firewall acts as a *man in the middle* for all TLS/SSL connections from the client computers to any external host. The firewall aggregates the *network flow* of all TCP/IP packets between a single client computer, client port, host IP address, and host port that result from a single HTTP request into an aggregated packet. This information is available for network devices that support the IPFIX [19] and NetFlow [20] formats. For each aggregated packet, 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, but not the URL string. This information is also passed to the threat analysis server. The threat analysis server processes the sequence of aggregated packets and may at any point in time issue an alarm, indicating that a particular client computer may be affected by malware.

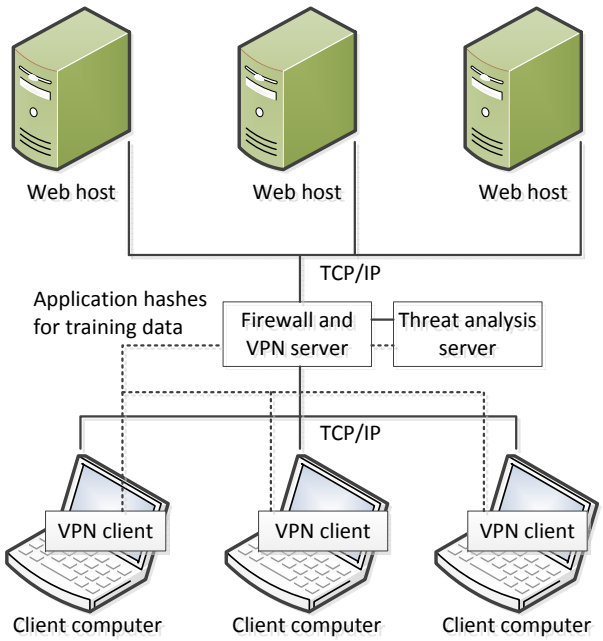

Fig. 1. System architecture

B. Operating Environment with Application Visibility

In the operating mode described in Section III-A, network flows of packets exchanged between a client computer and the web can generally originate from multiple applications, some of which may be malicious. We collect labeled training and evaluation data in an operating environment in which the firewall simultaneously acts as a *VPN server* and all client computers have VPN client software installed through which they access the network.

The VPN client software runs on each client computer and naturally has access to the processes which use the network interface. The VPN client identifies applications by means of a SHA hash code of their executable file, and communicates the association between TCP/IP packets and applications to the VPN server via a back channel. The VPN server can pass this information on to the threat-analysis server. For each packet, this hash code is stored in the log file. Hence, we can index the log file by the IP address of each client computer, the VPN user name of its user, and the hash key of the application that sent or received the packet.

TABLE I
NUMBER OF LABELED FLOWS AND PACKETS FOR EACH DATA SET

Data set	flows	pos. flows	neg. flows	packets	pos. packets	neg. packets
current data	579,064	1,625	577,439	23,043,756	98,376	22,945,380
future data	460,402	954	459,448	12,026,463	24,465	12,001,998

TABLE II
MALWARE TYPES.

Data Set	Malware Type	Flows	Packets
current data	Adware	497	50,708
current data	Trojans	967	43,398
current data	Potentially unwanted software	164	4,269
current data	Worms	1	1
future data	Adware	173	2,599
future data	Trojans	742	20,717
future data	Potentially unwanted software	38	1,143
future data	Worms	5	6

TABLE III
MOST FREQUENT APPLICATION NAMES OF MALWARE

Application	Flows	Packets
Browser extensions	767	14,055
Window drive manager	387	8,936
Ultrasurf	164	37,835
Client.exe	144	4,874
Premier opinion	114	5,926
UmmyVideoDownloader	12	3,275
Chrome	11	328
Sputnik FlashPlayer	8	321

C. Data Collection

We collect two different HTTPS data sets; we will refer to them as *current data* and *future data*, based on the roles which these data sets will play in our experiments. The *current data* contains the complete HTTPS traffic of 11 small to large computer networks that use the Cisco AnyConnect Secure Mobility Solution for a period of 7 days in February 2016. This data set contains 579,064 aggregated packets of 4,140 clients. We observe a total of 17,096 hashes.

The *future data* contains the complete HTTPS traffic of 10 small to large different computer networks that use the Cisco AnyConnect Secure Mobility Solution for a period of 7 days in April 2016. The data set contains 460,402 aggregated packets of 4,078 clients. Table I shows the number of total, positive (malicious) and negative (benign) network flows (sequences of aggregated packets) and packets for each data set. Table II enumerates the types and frequency of different types of malware, according to Virustotal.com. Table III enumerates the most frequent file names of applications that are classified as malicious by Virustotal.com.

D. Ground-Truth Labeling of HTTPS Traffic

The operating environment describes in Section III-B generates log files that contain, for each client IP address and VPN user name, sequences of aggregated TCP/IP packets that are associated with a hash key of the application that caused the network traffic. If we can obtain the malware status of an application that is identified by its hash key, we can label packets by whether their associated applications are malicious. Mapping signatures of executable files to a malware status indicator is the main functionality that virus scanning software provides.

Signature-based malware detectors can only detect malware after a particular file has *become known* to be malicious by some form of analysis. Since our goal is to obtain labels for the purposes of training and evaluating models for network traffic analysis, we can therefore record hashes and associated HTTPS traffic, and label the traffic in retrospect, after the malware status of most of the executable files has been established.

Virustotal.com is a web service that allows users to upload either executable files or MD5, SHA1, or SHA256 hashes of executable files. The files are run through a collection of 60 antivirus solutions, and the results of this analysis are returned. We upload the hash keys of all executable files that have generated 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 mode tools recognize the file as malicious, we label it as malicious. When only one or two virus scanners recognize the file as a virus, or when the hash key is unknown to Virustotal, we consider the malware status of the file *unknown*. We then label all traffic that has been generated by malicious executables as malicious, and all traffic of benign files as benign.

E. Client Malware Detection Problem

We will now establish the classification problem that the malware-detection model has to solve.

Our overall goal is to flag client computers that are hosting malware. Client computers are identified by a (local) IP address and a VPN user name, which allows us to distinguish between multiple devices that are used by a single user. For each client computer and each five-minute interval, the detection model receives a *network flow*—a sequence of aggregated TCP/IP packets $x_1, \ldots, x_T$. In addition to the client's IP address, each packet x_t has an observable host IP address, client and host ports, timestamp, duration (*i.e.,* the interval between the timestamps of first and last aggregated TCP/IP packet), and inbound and outbound data volume.

For each client computer and each five-minute interval, malware detection model f has to decide whether to raise an alarm. The model raises an alarm if the malware-detection score $f(x_1, \ldots, x_T)$ exceeds some threshold τ. In each five-

minute interval, model f with threshold τ will detect some proportion of malware-infected clients, and may raise false alarms for clients that do not actually host malware. The trade-off between the number of detections and false alarms can be controlled by increasing or decreasing threshold τ. Besides five-minute periods, we will also study long-term detections and false alarms. To this end, we will measure the number of infected and benign clients for which f with decision threshold τ raises an alarm over a period of seven days. We will measure the following performance metrics, both for an average 5-minute and an average 7-day period.

1) The following definitions depend on the concepts of *true positives*, *false positives*, and *false negatives*. The number n_{TP} of *true positives* is the number of malicious clients which are flagged by $f \geq \tau$ as malicious, the number n_{FP} is the number of benign clients that are flagged as malicious by $f \geq \tau$, and the number n_{FN} of *false negatives* is the number of malicious clients that are not flagged as malicious ($f < \tau$).
2) The *recall* $R = \frac{n_{TP}}{n_{TP}+n_{FN}}$ is the proportion of malicious applications that have created at least one packet in the reference period of five minutes or seven days and have been detected by $f \geq \tau$, relative to all (detected and undetected) malicious applications that have created at least one packet in the reference period.
3) The *precision* $P = \frac{n_{TP}}{n_{TP}+n_{FP}}$ is the proportion of malicious applications that have created at least one packet in the reference period of five minutes or seven days and have been detected by $f \geq \tau$, relative to all (malicious and benign) applications that have been flagged ($f \geq \tau$). Raising an alarm generally invokes some manual intervention by IT staff. The proportion of alarms that are false alarms is $1 - R$.
4) The recall at a specific precision, $R@x\%P$ quantifies the recall (proportion of malware that is detected) when the decision threshold is adjusted such that at most $1 - x\%$ of all alarms are false alarms.
5) The precision-recall curve of a decision function f shows the possible trade-offs that can be achieved by varying the decision threshold τ; decreasing τ tends to result in a higher recall and lower precision, increasing τ increases precision and decreases recall. The area under the precision-recall curve is an aggregate measure of the precision-recall curve; a higher area is better.

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 aggregated packet x_{it} is labeled by whether it has been sent or received by a malicious ($y_{it} = +1$) or benign ($y_{it} = -1$) application.

IV. HTTPS Network-Flow Analysis

This section derives our method that flags malware-infected client computers based on the analysis of their HTTPS network traffic. We will start by developing features on which the detection can be based, and then proceed to design a sequence classification model.

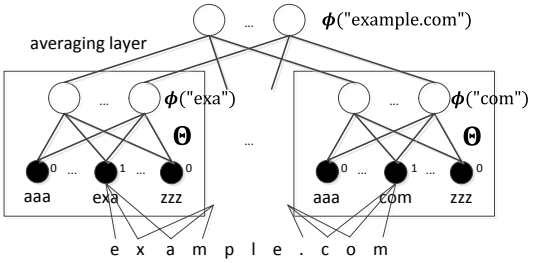

Fig. 2. Neural language model network architecture

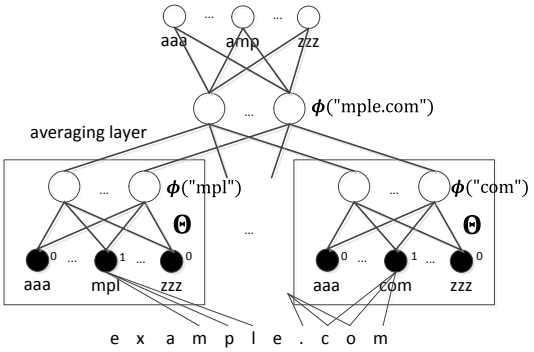

Fig. 3. Language model training

A. Packet Features

Our goal is to classify client computers as malware-infected or benign. Therefore, the detection model processes *network flows* of aggregated TCP/IP packets $x_1, \ldots, x_T$ sent to or received by one particular client computer. Each aggregated packet consists of client and host IP address, client and host ports, a timestamp, inbound and outbound data volume, and a duration. From each packet x_t, we extract a vector $\phi(x_t)$ that includes the one-hot encoded port value, duration, time gap from the preceding packet, the nnumbers of sent and received bytes, and the domain-name features that we will describe in Section IV-B.

B. Domain-Name Features

We explore several types of features that can be extracted from the host domain name.

1) Engineered Features: Franc et al. [8] develop a comprehensive set of 60 features of URL strings that can be used to detect malware by classifying HTTP traffic—for (unencrypted) HTTP traffic, the entire URL is visible to third parties. Their features include 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 of the string. We extract the vector of these *engineered* domain-name features for all domains.

2) 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-grams "exa", "xam", "amp", ..., ".co", and "com". The number of n-grams that occur in URLs grows almost exponentially in n; in our data, 1,583 character 2-grams, 54,436 character 3-grams, and 1,243,285 character 4-grams occur. If we added all character 3-gram features to the feature representation $\phi(x_t)$ of a packet, then the total number of features of an entire network flow of T packets would be prohibitively large; it would impose a heavy computational burden, and cause overfitting of the malware-detection model. In our experiments, we therefore explore character 2-gram features.

3) Neural Domain-Name Features: We condense the set of character n-grams which a domain string is composed of into a low-dimensional representation by means of a *neural language model*. Neural language models [10] derive low-dimensional, continuous-state representations of words which have the property that words which tend to appear in a similar context have a similar representation. Here, "words" are the overlapping character n-grams that constitute a domain name. We apply neural language models with the goal of finding a representation such that substrings that tend to co-occur in URL strings have a similar vector representation.

We use the following neural-network architecture, illustrated in Figure 2. 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; Figure 2 illustrates the case of $n = 3$. The input layer is fully connected to a hidden layer with weight matrix Θ; we vary the number k of hidden units in our experiments. 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 be able to complete an initial part of a sentence, which forces the network to find a vector-space representation of words 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—the top-level domain—and starts with the most specific subdomain. While, in most cases, language models are trained to predict the next word given a window of preceding words, here we reverse this processing order. During training, the network processes a sliding window of m overlapping n-grams in parallel, and averages the m resulting activation vectors of hidden units; Figure 3 illustrates the case of $m = 6$, $n = 3$, and an input window of "mpl", ..., "com". The hidden layer is connected to a soft-max output layer that again has one dimension for each possible character n-gram in a URL string. The output units infer a probability distribution over character n-grams. The output target for the output layer during training is the character n-gram in the URL string that immediately precedes the input window of m overlapping n-grams; in the example of Figure 3, the target value is "amp".

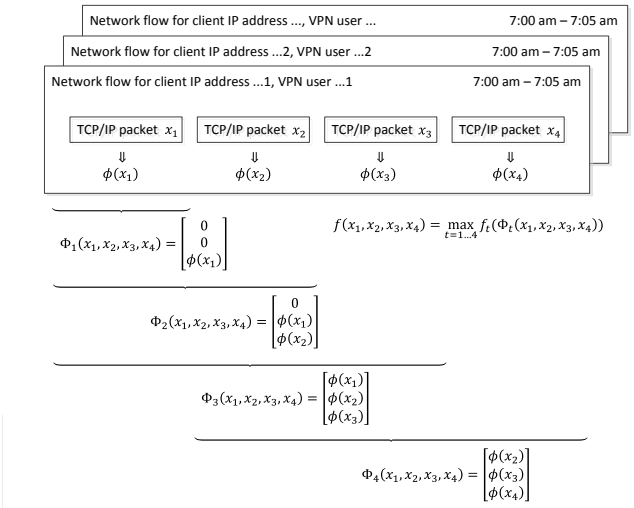

Fig. 4. Client classifier

We train the neural language model using all URLs that occur in our data. In addition, we add the 500,000 domains that have the highest web traffic according to alexa.com to the training data. For each URL and each position of the sliding window, we perform a back-propagation update step for which the character that immediately precedes the input window serves as prediction target. We use the word2vec software package [21]. The neural language model generates a set of k host-URL features per aggregated packet.

C. Client Classifier

The client classifier is visualized in Figure 4. At application time, the *client classifier* processes a client computer's network flow by moving a sliding window of width w over the sequence of packets. For each position t, feature vector $\Phi_t(x_1, \ldots, x_T) = [\phi(x_{t-w+1}) \ldots \phi(x_t)]^\top$ stacks the feature vectors of the w aggregate packets x_{t-w+1} to x_t; feature vectors with zero or negative index are filled with zeros. Feature vector $\Phi_t(x_1, \ldots, x_T)$ serves as input to decision function f_t. The decision-function value is maximized over all window positions:

$$f(x_1, \ldots, x_T) = \max_{t=1}^{T} f_t(\Phi_t(x_1, \ldots, x_T)), \tag{1}$$

and the client from which the network flow originates is classified as malware-infected if its decision-function value exceeds the decision threshold:

$$\hat{y} = \begin{cases} +1 & \text{if } f(x_1, \ldots, x_T) \geq \tau \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

Function f_t is a standard decision function of a classification problem with vector-valued input.

Training data $S = \bigcup_{i=1}^{n} \{((x_{i1}, y_{i1}), \ldots, (x_{iT_i}, y_{iT_i})\}$ labels each packet by whether it has been sent or received by a

malicious ($y_{it} = +1$) or benign ($y_{it} = -1$) application. In order to train a model f_t, the data is transformed as follows. For each network flow i in the training data and each window position t, a training example $(\Phi_t(x_{i1}, \ldots, x_{iT}), \bar{y}_{it})$ is created, where $\bar{y}_{it} = \max_{j=t-w+1}^{t} y_{ij}$; that is, the feature representation of network flow i and window position t is $+1$ if any packet in this window position is associated with a malicious application. This process creates a sample $\bar{S} = \bigcup_{i=1}^{n} \bigcup_{t=1}^{T_i} \{(\Phi_t(x_{i1}, \ldots, x_{iT_i}), \bar{y}_{it})\}$ that is of size n times the average sequence length. Sample $\bar{S}$ serves as input to a standard classification learning procedure.

In our empirical study, we have tens of millions of training instances in $\bar{S}$. We can therefore rule out kernel methods, because the kernel matrix would require in the order of 10^{14} floating-point values that have to be stored in memory. Based on preliminary experimentation, we implement decision function f_t as a random-forest classifier. We also include a linear classification model in the experiments as a baseline.

V. Experiments

A. Classification of Host Domains

In our first experiment, we investigate the type of domain-name features—engineered, neural, and character 2-grams—with respect to their ability to distinguish between domains that are contacted by benign versus malicious applications, and we tune the hyper-parameters of the neural domain-name language model. 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 data, 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"). For 90,445 of the domains, only the IP address string is available because the client request contained the IP address instead of a domain name.

We infer engineered domain-name features, character 2-grams, and the vector-space representation of each domain string using the neural language model, as described in Section IV-B. 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 IV shows the area under the precision-recall curve for the different set of feature types. We find that a parameter combination of $n = 6$ (input character 6-grams), $m = 4$ (during training, the vector-space representation of 4 adjacent character 6-grams is averaged) and $k = 100$ (the vector-space representation of a domain name has 100 dimensions) works best. We vary the parameters individually; Table IV shows the resulting area under the precision-recall curve. We compare a version of the neural language model which is trained to predict the preceding window of n-grams—as described in Section IV-B—and a version that is trained "from left to right", to predict the succeeding n-gram. Training the neural network in the natural reading order for domain strings—from right to left—works best. Comparing the neural domain features to the raw character 2-gram end the 60 engineered features, 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.

TABLE IV
Domain classification using various domain-name features

Feature type	parameters	Area under PR curve
neural	$n = 4$	0.37
neural	$n = 5$	0.38
neural	$n = 6$	**0.39**
neural	$n = 7$	0.38
neural	$k = 50$	0.38
neural	$k = 100$	**0.39**
neural	$k = 200$	0.38
neural	$m = 3$	0.35
neural	$m = 4$	**0.39**
neural	$m = 5$	0.33
neural	left to right	0.23
neural	right to left	**0.39**
char 2-grams		0.24
engineered		0.32
neural+engineered		0.36

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 V shows the highest-scoring domains. They include IP addresses without DNS entries, cloud services, subdomains of the YouTube and Facebook content delivery networks, and domains that do not host visible content and have most likely been registered for malicious purposes.

B. Detection of Current Malware

This section reports on experiments in which training and test data are sampled from the *current data set*. We explore the contribution of several feature types. We split the client computers—identified by a pair of a local IP address and a VPN user name—into 4 partitions, and conduct 4-fold cross validation over these partitions. This ensures that each user and client computer is either in the training part, or else in the evaluation part of the data. As a side note, when we allow client computers into the test set for which earlier network flows are included in the training set, we can classify

TABLE V
DOMAIN-NAMES MOST CONFIDENTLY CLASSIFIED AS MALICIOUS

Domain
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

TABLE VI
SLIDING-WINDOW WIDTH w OF THE CLIENT CLASSIFIER

Width w	**R@70%P**	**R@80%P**	**R@90%P**
1	0.291 ± 0.162	0.228 ± 0.164	0.159 ± 0.161
2	0.352 ± 0.151	0.227 ± 0.211	0.129 ± 0.129
3	0.396 ± 0.126	0.305 ± 0.147	0.182 ± 0.105
4	**0.411 ± 0.115**	**0.349 ± 0.092**	**0.237 ± 0.038**
5	0.380 ± 0.098	0.325 ± 0.051	0.223 ± 0.050
6	0.358 ± 0.083	0.281 ± 0.053	0.214 ± 0.067
7	0.377 ± 0.099	0.299 ± 0.054	0.206 ± 0.065
8	0.355 ± 0.071	0.262 ± 0.089	0.204 ± 0.078
8	0.367 ± 0.081	0.272 ± 0.095	0.170 ± 0.084

these clients nearly perfectly. We run a nested inner loop of twofold cross validation in which we tune the parameters of the random forest classifier and the regularization parameter of the linear model by grid search. The grid spans over a maximum tree depth of 3 or unlimited depth, and a maximum number of features, minimum number of instances for a split, and minimum number of instances in a leaf of 1, 3, and 10. We find the optimal combination in the inner cross-validation loop, then train a model using these settings on the entire training portion and evaluate that model in the outer cross-validation layer. We use a fixed number of 1,000 trees.

We then split all training and test sequences into blocks of five minutes. The client classifier that we described in Section IV-C processes a sliding window of w packets in each step. In our first experiment, we explore different values of w. Table VI shows that using packet features, a window width of $w = 4$ performs best. In additional experiments, we observe that the optimal width lies between 2 and 5 for any set of features. We fix $w = 4$ for all subsequent experiments.

We will now study the accuracy of the client classifier and the contributions of the various domain-name and packet features. We train a random-forest sequence classifier with a sliding-window width of 4 on all five-minute blocks of the training-part of the data and apply the model to each five-minute block of the test data.

Table VII shows the recall rates for various precision values of the random-forest classifier. The numerical packet features allow to detect 40% of malware-infected clients at a precision of 70% or 25% at a precision of 90%. Domain-name character 2-gram features do not improve the detection performance compared to just flow features. Using character n-gram features for higher values of n is computationally not feasible. By contrast, a combination of packet features and neural domain-name features leads to a substantial improvement over just packet features; over a 5-minute interval, it detects 58% of malware-infected clients at a precision of 70%. That is, 30% of all alarms are false alarms. At a higher decision threshold, it detects 44% of infected clients at a precision of 90% (with 10% of the alarms being false alarms).

We compare the random-forest classifier to a linear classification baseline. Using the packet and neural domain-name features, an ℓ_2-regularized *logistic regression classifier* attains a recall of 17.73% at a precision of 70%, a recall of 16.20% at a precision of 80%, or a recall of 11.51% at a recall of 90%. We conclude that the random-forest classifier substantially outperforms the linear baseline and exclude the logistic regression model from all further experiments.

We measure the random-forest classifier's ability to identify malware-infected clients over a period of seven days. We have the model produce an output for every client and every 5-minutes interval over a period of seven days. In this setting, a true positive is a malicious client that is flagged by f at least once, a false positive is a benign client that is flagged by f at least once, and a false negative is a malicious client that is not flagged over the entire observation period of seven days. Table VIII shows the recall at various precision values. The relative merit of neural domain features, packet features, and domain-name character n-gram features is the same, but the overall recognition rates are higher. Using a combination of numeric packet features and neural domain-name features the client classifier detects 78% of all malware-infected clients at a precision of 70% or 60% malware-infected clients at a precision of 90% within seven days.

C. Detection of Future Malware

In Section V-B, the training and test data are governed by identical distributions. In practice, however, a malware detection model is deployed and used while new emerges. This section studies the robustness of the client classifier on future TCP/IP traffic.

We use the entire *current data set* as training data and the *future data set* as test data. We run twofold cross validation on the training data to we tune the parameters of the random forest classifier by grid search, as in Section V-B. We train a model with tuned parameters on the entire training data and evaluate that model on the more recent test data.

We first measure the performance over 5-minute intervals. Table IX shows the resulting recall rates for various precision values. Most noticably, the performance of the packet features has deteriorated sharply. The combination of numeric packet features and neural domain-name features still performs best and still detects 44% of all malware at a precision of 70% or 38% at a precision of 90%. Table X shows the recall at various precision values over the entire test period of seven days. Again, the packet features have deteriorated substantially. But the combination of packet features and neural domain-name

TABLE VII
CLIENT CLASSIFIERS ON 5-MINUTE INTERVALS OF CURRENT TRAFFIC

Feature	R@70%P	R@80%P	R@90%P	Area under PR curve
packet	39.88 ± 5.684	36.18 ± 5.444	24.84 ± 2.315	0.461 ± 0.040
neural	20.49 ± 4.146	13.63 ± 2.764	6.806 ± 1.388	0.317 ± 0.044
packet + neural	**58.00 ± 3.527**	**52.15 ± 5.156**	**44.06 ± 7.900**	**0.606 ± 0.029**
packet + engineered	48.37 ± 6.551	42.21 ± 7.231	24.97 ± 4.383	0.526 ± 0.032
packet + 2-gram	36.36 ± 8.012	32.80 ± 7.878	20.14 ± 5.127	0.431 ± 0.054

TABLE VIII
CLIENT CLASSIFIERS ON 7 DAYS OF CURRENT TRAFFIC

Feature	R@70%P	R@80%P	R@90%P	Area under PR curve
packet	47.82 ± 8.421	34.58 ± 12.15	23.27 ± 9.570	0.590 ± 0.049
neural	29.92 ± 1.359	19.94 ± 0.890	9.959 ± 0.453	0.505 ± 0.019
packet + neural	**77.75 ± 1.198**	**66.01 ± 3.991**	**37.56 ± 3.770**	**0.765 ± 0.015**
packet + engineered	55.00 ± 8.288	38.41 ± 6.987	19.16 ± 3.503	0.650 ± 0.038
packet + 2-gram	41.79 ± 5.113	27.82 ± 3.408	13.88 ± 1.715	0.581 ± 0.036

TABLE IX
CLIENT CLASSIFIERS ON 5-MINUTE INTERVALS OF FUTURE TRAFFIC

Feature type	R@70%P	R@80%P	R@90%P	Area under PR curve
packet	11.01	0.900	0.400	0.194
neural	15.71	10.41	5.205	0.251
packet + neural	**43.54**	**41.84**	**37.83**	**0.459**
packet + engineered	39.33	32.13	15.81	0.399
packet + 2-gram	29.42	26.22	15.51	0.340

TABLE X
CLIENT CLASSIFIERS ON 7 DAYS OF FUTURE TRAFFIC

Feature type	R@70%P	R@80%P	R@90%P	Area under PR curve
packet	22.82	7.507	3.703	0.435
neural	32.83	21.92	10.91	0.524
packet + neural	**79.47**	**74.27**	**70.47**	**0.806**
packet + engineered	73.47	59.45	29.72	0.713
packet + 2-gram	54.35	36.23	18.11	0.631

features have maintained their predictive power and actually perform better on future data than they did on current data. In this experiment, the classifier has been trained with slightly more training data because none of the clients in the *current data set* has to be held back for evaluation. This explains how the model can perform better then the model that is evaluated on current data.

We investigate the sharp deterioration of the effectiveness of the numerical packet features. We find that the average duration of aggregated malicious packets is much lower in the future than in the current data set. Also, the proportion of malicious packets with low outgoing data volume is higher in the future data set, conversely, the proportion of high-volume incoming benign packets is higher in the future than in the current data set. We cannot identify particular types of malware or individual benign applications as being the source of this distributional shift. We have to conclude that as the availability and overall use of software changes, distributional properties of TCP/IP traffic are nonstationaty.

D. Detection of Previously Unknown Malware

Finally, we specifically study the performance on novel malware hashes that did not occur in the training data. For these experiments, we train the models on the *current data set* as in Section V-C. We evaluate the models on the *future data set*, but we measure the recall only on client computers that host malicious applications whose hash does not occur in the training period. That is, any non-infected client that is flagged counts as a false positive, but only clients that host malware whose application hashes do not occur in the training data counts as true positive.

Again, we first measure the performance over 5-minute intervals. Table XI shows the recall rates for various precision values. In this setting, the packet features in isolation are not sufficient to identify any malware at a precision of 70% or

higher. However, the packet features still carry information about the malware status, because the combination of packet and neural domain-name features performs substantially better than the neural domain-name features in isolation. Again, neural domain-name features outperform engineered and 2-gram domain-name features. Based on packet and neural domain-name features, the client classifier is able to detect 29% or all *previously unknown* malware at a precision of 70% or 31% at a precision of 90%.

We next study the performance over periods of seven days. Table XII shows that based on the combination of packet and neural domain-name features, the client classifier detects 62% of all malware at 70% precision or 44% malware at 90% precision within seven days. We would like to stress that the evaluation is performed only on new, unknown malware that has first been observed after the classifier has been trained. A signature-based detection method cannot detect any of these malware instances.

We analyze the classifier's ability to detect previously unknown malware applications in depth. We find that of the overlap between domains that are contacted by unknown malware applications in the *future data set* and domains that are contacted by malware in the *current data set* is only 37%. We next focus on the restricted set of client computers in the *future data set* that only host malware that is *unknown* (file hash does not occur in the *current data*) and does *not contact any known domain* (which occurs in the *current data*). That is, we focus on malware for which neither the application hash nor any contacted domain occurs in the training data. We find that the client classifier still flags 40% of these 27 clients as malicious at a precision of 70%. This shows that the client classification model is able to identify even unknown malware based on characteristics of the TCP/IP packets and generalized features of the domain name, rather than relying on specific TCP/IP traffic patterns of known malware and contact to known malicious domains.

VI. Conclusion

We have studied malware detection in client computers by HTTPS traffic analysis. In order to collect labeled training data, we have designed an environment in which a VPN client associates all network flows with the executable file that has caused the traffic. We query the file hashes to Virustotal.com in order to obtain TCP/IP network flows which are associated to known malicious and benign software. HTTPS traffic offers very little information, because the entire payload—including the URL—is encrypted. In order to extract as much information as possible from the host IP address, we employ a neural language model that transforms the domain-name string into a continuous-space representation. We devise a classifier that processes packet and domain-name features of a sliding window of TCP/IP packets.

We can draw a number of conclusions from our experiments. Neural domain-name features consistently outperform engineered domain-name features and character n-grams. In all settings, the combination of numerical packet features (timestamps, incoming and outgoing data volumes) and neural domain-name features gives the best performance. We find empirically that the client classifier that processes a sliding window of packet and neural domain-name features can identify malware with a high precision (*e.g.*, with 80% recall at 90% precision) both on current network traffic and on network traffic that has been observed two months after the classifier has been trained. Perhaps most notably, the model is able to identify more than 60% of new, unknown malware whose executable file has first been observed after the model has been trained at 70% precision, and more than 40% at 90% precision.

In our operating environment, the traffic analysis is performed within dedicated network equipment rather than on the client computer. In this setting, the threat-analysis system protects all clients in the network, independent of any antivirus software which they may additionally be running. Therefore, the executable file is not accessible by the analysis tool and neither static nor dynamic code analysis can be carried out. However, our experiments highlight the importance of network-traffic analysis an orthogonal approach: based on the analysis of HTTPS traffic, the majority of malware with new, previously unknown executable files can still be detected.

Acknowledgment

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

References

[1] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *Journal in Computer Virology*, vol. 1, no. 1-2, pp. 13–23, 2005.

[2] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, p. 2006, 2006.

[3] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, pp. 56–64, 2014.

[4] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX Conference on Security*, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028070

[5] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," in *Proceedings of the IEEE Conference on Security and Privacy*, vol. 5, 2007, pp. 32–39.

[6] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the Annual Network and Distributed System Security Symposium*, 2008.

[7] R. Perdisci, W. Lee, and N. Feamster, "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.

[8] V. Franc, M. Sofka, and K. Bartos, "Learning detector of malicious network traffic from weak labels," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 85–99.

[9] T. I. Movement, "SSL Pulse," 2016, https://www.trustworthyinternet.org/ssl-pulse/.

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[11] K. Bartos and M. Sofka, "Robust representation for domain adaptation in network security," in *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 116–132.

[12] T. Nelms, R. Perdisci, and M. Ahamad, "Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates." in *Proceedings of the USENIX Security Symposium*, 2013, pp. 589–604.

TABLE XI
CLIENT CLASSIFIERS ON 5-MINUTE INTERVALS OF TRAFFIC FROM UNKNOWN HASHES

Feature type	R@70%P	R@80%P	R@90%P	Area under PR curve
packet	0.0	0.0	0.0	0.045
neural	6.906	4.604	2.302	0.133
packet + neural	**28.92**	**25.52**	**23.32**	**0.313**
packet + engineered	14.71	9.609	4.804	0.244
packet + 2-gram	9.009	6.006	3.003	0.168

TABLE XII
CLIENT CLASSIFIERS CLASSIFIERS ON 7 DAYS OF TRAFFIC FROM UNKNOWN HASHES

Feature type	R@70%P	R@80%P	R@90%P	Area under PR curve
packet	0.700	0.500	0.200	0.143
neural	27.42	18.31	9.109	0.456
packet + neural	**61.56**	**58.25**	**43.54**	**0.686**
packet + engineered	44.34	29.52	14.71	0.563
packet + n-gram	28.12	18.71	9.309	0.484

[13] J. Kohout and T. Pevny, "Unsupervised detection of malware in persistent web traffic," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2015, pp. 1757–1761.

[14] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, 2006.

[15] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 5–16, 2007.

[16] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Computer Networks*, vol. 53, no. 1, pp. 81–97, 2009.

[17] J. Kohout and T. Pevny, "Automatic discovery of web servers hosting similar applications," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, 2015, pp. 1310–1315.

[18] J. Lokoč, J. Kohout, P. Čech, T. Skopal, and T. Pevnỳ, "k-nn classification of malware in https traffic using the metric space approach," in *Intelligence and Security Informatics*. Springer, 2016, pp. 131–145.

[19] B. Claise, B. Trammell, and P. Aitken, "Specification of the ip flow information export (ipfix) protocol for the exchange of flow information," 2013, https://tools.ietf.org/html/rfc7011.

[20] Cisco Systems, "Cisco ios netflow, http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html," 2016.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.