Machine Learning (2019) 108:1353–1368
https://doi.org/10.1007/s10994-019-05789-z

Joint detection of malicious domains and infected clients

Paul Prasse[1] · René Knaebel[1] · Lukáš Machlica[2] · Tomáš Pevný[2,3] · Tobias Scheffer[1]

Received: 6 September 2018 / Accepted: 25 January 2019 / Published online: 25 February 2019

Abstract
Detection of malware-infected computers and detection of malicious web domains based on their encrypted HTTPS traffic are challenging problems, because only addresses, timestamps, and data volumes are observable. The detection problems are coupled, because infected clients tend to interact with malicious domains. Traffic data can be collected at a large scale, and antivirus tools can be used to identify infected clients in retrospect. Domains, by contrast, have to be labeled individually after forensic analysis. We explore transfer learning based on sluice networks; this allows the detection models to bootstrap each other. In a large-scale experimental study, we find that the model outperforms known reference models and detects previously unknown malware, previously unknown malware families, and previously unknown malicious domains.

Keywords Machine learning · Neural networks · Computer security · Traffic data · Https traffic

Editors: Karsten Borgwardt, Po-Ling Loh, Evimaria Terzi, Antti Ukkonen.

✉ Paul Prasse
prasse@cs.uni-potsdam.de

René Knaebel
knaebel@cs.uni-potsdam.de

Lukáš Machlica
lumachli@cisco.com

Tomáš Pevný
tpevny@cisco.com

Tobias Scheffer
scheffer@cs.uni-potsdam.de

[1] Department of Computer Science, University of Potsdam, Potsdam, Germany

[2] Cisco R&D, Prague, Czech Republic

[3] Department of Computer Science, Czech Technical University in Prague, Prague, Czech Republic

1 Introduction

Malware violates users' privacy, harvests passwords and personal information, can encrypt users' files for ransom, is used to commit click-fraud, and to promote political agendas by popularizing specific content in social media (Kogan 2015). Client-based antivirus tools use vendor-specific blends of signature-based analysis, static analysis of portable-executable files, emulation (partial execution without access to actual system resources prior to execution in the actual operating system) and dynamic, behavior-based analysis to detect malware (Swinnen and Mesbahi 2014). Network-traffic analysis complements antivirus software and is widely used in corporate networks. Traffic analysis allows organizations to enforce acceptable-use and security policies consistently throughout the network and minimize management overhead. Traffic analysis makes it possible to encapsulate malware detection into network devices or cloud services that can detect polymorphic malware (Karim et al. 2005) as well as yet-unknown malware based on, for instance, URL patterns (Bartos and Sofka 2015).

However, malware can easily prevent the analysis of its HTTP payload by using the encrypted *HTTPS* protocol. The use of HTTPS by itself is not conspicuous because Google, Facebook, LinkedIn, and many other popular sites encrypt their network traffic by default and the global data volume of HTTPS has surpassed that of HTTP (Finley 2017). In order to subject HTTPS traffic to network-traffic analysis, organizations today have to configure their network such that all web traffic is routed via a web-security server. This server's root certificate has to be installed as a trusted certificate on all client computers, which allows the service to act as a man-in-the-middle between client and host. It can decrypt, inspect, and re-encrypt HTTPS requests. This approach scales poorly to large networks because the cryptographic operations are computationally expensive, and it introduces a potential vulnerability into the network.

Without breaking the encryption, an observer of HTTPS traffic can only see the client and host IP addresses and ports, and the timestamps and data volumes 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. The HTTP payload, including the HTTP header fields and the URL, are encrypted.

Web hosts are involved in a wide range of illegitimate activities, and blacklisting traffic to and from known malicious domains and IP addresses is an effective mechanism against malware. Malicious domains can host back-ends for banking trojans and financial scams, click-fraud servers, or distribution hubs for malicious content. Identifying a domain as malicious requires a complex forensic analysis. An analyst has to collect information about the server that hosts the domain, software and employed technologies, and can research ownership of the domain and co-hosted domains as well as observe the host's behavior.

Since many types of malicious activities involve interaction with client-based malware, the detection of malicious hosts and infected clients are coupled problems. In the context of neural networks, labeled data for related tasks are often exploited by designing coupled networks that share part of the parameters. In sluice networks (Ruder et al. 2017), the extent to which parameters are shared is itself controlled by parameters, which allows auxiliary data to serve as a flexible prior for the task at hand.

The rest of this paper is structured as follows. Section 2 reviews related work. We describe our operating environment and our data in Sect. 3 and the problem setting in Sect. 4. In Sect. 5, we derive a model for joint detection for malware and malicious domains and describe reference methods. Section 6 presents experiments; Sect. 7 concludes.

2 Related work

Prior work on the analysis of *HTTP logs* (Nguyen and Armitage 2008) has addressed the problems of identifying command-and-control servers (Nelms et al. 2013), unsupervised detection of malware (Kohout and Pevny 2015b; Bartos et al. 2016), and supervised detection of malware using domain blacklists as labels (Franc et al. 2015; Bartos and Sofka 2015). HTTP log files contain the full URL string, from which a wide array of informative features can be extracted (Bartos and Sofka 2015).

A body of recent work has aimed at detecting Android malware by network-traffic analysis. Arora et al. (2014) use the average packet size, average flow duration, and a small set of other features to identify a small set of 48 malicious Android apps with some accuracy. Lashkari et al. (2015) collect 1500 benign and 400 malicious Android apps, extract flow duration and volume feature, and apply several several machine-learning algorithms from the Weka library. They observe high accuracy values on the level of individual flows. Demontis et al. (2018) model different types of attacks against such detection mechanisms and devise a feature-learning paradigm that mitigates these attacks. Malik and Kaushal (2016) aggregate the VirusTotal ranking of an app with a crowd-sourced domain-reputation service (Web of Trust) and the app's resource permission to arrive at a ranking.

Prior work on *HTTPS logs* has aimed at identifying the application layer protocol (Wright et al. 2006; Crotti et al. 2007; Dusi et al. 2009). In order to cluster web servers that host similar applications, Kohout and Pevny (2015a) develop features that are derived from a histogram of observable time intervals and data volumes of connections. Using this feature representation, Lokoč et al. (2016) develop an approximate k-NN classifier that identifies servers which are contacted by malware. Hosts that are contacted by malware are by no means necessarily malicious. Malware uses URL forwarding and other techniques to route its traffic via legitimate hosts, and may contact legitimate services just to dilute its network traffic. We will nevertheless use the histogram features as a reference feature representation.

Graph-based classification methods (e.g., Anderson et al. 2011) have been explored but cannot be applied in our operating environment. In our operating environment, a Cloud Web Security server observes only the network traffic within an organization. In order to perceive a significant portion of the network graph, companies would have to exchange their network-traffic data which is impractical for logistic and privacy reasons.

Prior work on neural networks for network-flow analysis (Pevny and Somol 2016) 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. Malware detection from HTTPS traffic has been studied using a combination of word2vec embeddings of domain names and long short term memory networks (LSTMs) (Prasse et al. 2017). We will use this method as a reference in our experiments. Recent findings suggest that the greater robustness of convolutional neural networks (CNNs) outweighs the ability of LSTMs to account for long-term dependencies (Gehring et al. 2017). This motivates us to explore convolutional architectures. Neural networks have also been applied to static malware analysis (Pascanu et al. 2015).

In the context of deep learning, multi-task learning is most often implemented via hard or soft parameter sharing of hidden layers. In hard parameter sharing, models for all task can share the convolutional layers (Long and Wang 2015) or even all hidden layers (Caruana 1993), which can dramatically increase the sample size used to optimize most of the parameters (Baxter 1997). Soft parameter sharing, by contrast, can be realized as a direct application of hierarchical Bayesian modeling to neural network: each parameter is regular-

ized towards its mean value across all tasks (Duong et al. 2015; Yang and Hospedales 2016). Cross-stitch (Misra et al. 2016) and sluice networks (Ruder et al. 2017) allow the extent of task coupling for separate parts of the network to be controlled by parameters. Sluice networks have a slightly more general form than cross-stitch networks because they have additional parameters that allow a task-specific weighting of network layers.

Alternative transfer-learning approaches for neural networks enforce an intermediate representation that is invariant across tasks (Ganin et al. 2016). Outside of deep learning, the group lasso regularizer enforces subspace sharing, and wide range of approaches to multi-task learning have been studied, based on hierarchical Bayesian models (e.g., Finkel and Manning 2009), learning task-invariant features (e.g., Argyriou et al. 2007), task-similarity kernels (Evgeniou et al. 2005), and learning instance-specific weights (e.g., Bickel et al. 2008).

3 Operating environment

This section describes our application 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, and all external HTTP and HTTPS connections from any client within the organization is then routed via this service. The service can block HTTP and HTTPS requests based on the host domain and on the organization's acceptable-use policy. The CWS service blocks all traffic to and from all malicious domains on a curated blacklist. It issues warnings when it has detected malware on a client. Since security analysts have to process the malware warnings, the proportion of false alarms among all issued warnings has to be small.

On the application layer, HTTPS uses the HTTP protocol, but all messages are encrypted via the Transport Layer Security (TLS) protocol or its predecessor, the Secure Sockets Layer (SSL) protocol. 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*. For each network flow, a line is written into the log file that includes data volume, timestamp, client and host address, 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 (Blum and Lueker 2001), this server caches DNS request-response pairs and can map IP addresses to previously resolved domain names. The resulting sequence of log-file lines serves as input to the detection models for malware and malicious domains.

3.1 Data collection

For our experiments, we combine a large collection of HTTPS network flows (Prasse et al. 2017) that have been labeled by whether they originate from a malicious or legitimate application with a domain blacklist that is maintained by forensics experts at Cisco.

Table 1 Key statistics of the HTTPS network-traffic data sets

Data set	Flows	Malicious	Benign	Users	Infected	Organizations
Training	44,348,879	350,220	43,150,605	133,437	8944	171
Test	149,005,149	955,037	142,592,850	177,738	8971	169

Table 2 Number of applications in HTTPS network-traffic data sets

Data set	Applications	Malicious
Training	20,169	1168
Test	27,264	1237

Prasse et al. (2017) have collected the HTTPS network flows that pass CWS servers in 340 corporate networks. The client computers in these networks run a VPN client that monitors the process table and network interface, and keeps a record of which executable file creates each network flow. In retrospect, the executable files have been analyzed with a multitude of antivirus tools. The resulting data set consists of network flows between known clients (identified by organization and VPN account), domains (fully qualified domain names), data volumes and timestamps, and a label that indicates whether the application that generated the traffic is recognized as malware by antivirus tools. We stratify training and test data in chronological order. The *training data* contains the complete HTTPS traffic of 171 small to large computer networks for a period of 5 days in July 2016. The *test data* contains the complete HTTPS traffic of 169 different computer networks for a period of 8 days in September 2016. Forensics experts at Cisco continuously investigate suspicious host names, second-level domain names, and server IP addresses that have been flagged by a wide range of mechanisms. This includes an analysis of the hosted software and employed technologies, of registry records, URL and traffic patterns, and any additional information that may be available for a particular domain. We believe that domains are almost never erroneously rated as malicious, but due to the expensive analytic process, the blacklist of malicious domains is necessarily incomplete. All traffic from and to malicious serves can easily be blocked by the CWS service. The network traffic does not contain any flows to domains that had been on our blacklist at the time when the traffic data were collected. The traffic data set contains network flows to and from 4340 malicious host names, second-level domains, and server IP addresses that have been added to the blacklist after the data were collected.

3.2 Quantitative analysis of the data

Tables 1 and 2 summarizes the number of benign and malicious network flows, client computers, infected computers, applications with unique hashes, and organizations.

Table 3 gives statistics about the most frequent malware families. It enumerates the number of variations that occur, the number of infected clients, and, in parentheses, the number of infected clients in the training data.

In total, just below 18,000 computers are malware-infected and communicate with domains that had not been blacklisted at the time, which corresponds to almost 0.6%.

In the traffic data, 4340 domains occur that have been added to the blacklist after the traffic data were recorded. Table 4 details the types of malicious host names, second-level domains, and server IP addresses that occur in all data and in the training data.

Table 3 Malware families and malware types

Malware family	Variations	Clients
Dealply	506	1385 (516)
Softcnapp	119	797 (250)
Crossrider	98	274 (102)
Elex	86	779 (316)
Opencandy	57	164 (126)
Conduit	56	314 (103)
Browsefox	52	78 (34)
Speedingupmypc	29	224 (63)
Kraddare	28	33 (26)
Installcore	27	49 (19)
Mobogenie	26	467 (184)
Pullupdate	25	99 (25)
Iobit downloader	24	38 (15)
Asparnet	24	5267 (5128)

Table 4 Domain-label statistics

Type	Total	Training
Malware-distribution	2730	478
Ad-injector	961	576
Malicious-content-distribution	276	171
Potentially unwanted application	97	75
Click-fraud	65	50
Spam-tracking	61	51
Information-stealer	52	22
Scareware	30	23
Money-scam	22	9
Banking-trojan	19	10
Malicious-advertising	13	10
Cryptocurrency-miner	9	0
Ransomware	3	3
Anonymization-software	2	1

4 Problem setting

We will now establish the problem setting. Our goal is to flag client computers that are hosting malware, and to flag malicious web domains. Client computers are identified by a (local) IP address and a VPN user name; web domains are identified by a fully qualified domain name or, when no domain name can be observed, an IP address.

We have two types of classification instances. For each interval of 24 hours, we count every client computer that establishes at least one network connection as a separate *classification instance* of the malware-detection problem. A client that is active on multiple days constitutes multiple classification instances; this allows us to issue daily infection warnings for clients. Equivalently, for each interval of 24 hours, we model each observed fully qualified domain

name as a classification instance. This allows us to make daily blacklisting decisions, and to disregard traffic after 24 hours in the deployed system.

Our training data are labeled at the granularity of a network flow between a client and a host. This allows us to train classification models at the granularity of network flows. Learning a network-flows classifier from labeled flows is an intrinsically easier problem than learning a detection model from labels at the granularity level of clients or domains. While a detection model that is trained from labeled clients or domains has to figure out which flows pertain to the malicious activity, the network-flow classification model is handed that information during training.

Since a client is infected if it is running at least one malicious application and a domain is malicious if it engages in at least one malicious activity, it is natural to aggregate the classification results for network flows into detection results for clients and domains by max-pooling the decision-function values over all flows for that client or domain, respectively, throughout the period of 24 hours. The flow classifiers are thereby applied as one-dimensional convolutions over time; max-pooling the outcome yields detection models for infected clients and malicious domains. Since an application generally generates multiple network flows, it may be helpful to take the context into account when classifying each flow. Our input representation therefore includes a window of the client's flows that is centered over the flow to be classified. The width of this window is a model parameter. This window always contains the context of network flows for a client, both for detection of malware and of malicious domains. While the CWS server can observe the complete traffic of each client in the network, it will generally only observe a small fraction of traffic to and from a domain outside the network.

We will measure precision-recall curves because they are most directly linked to the merit of a detection method from an application point of view. Precision—the fraction of alarms that are not false alarms—is directly linked to unnecessary workload imposed on security analysts, while recall quantifies the detection rate. 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. Note that the relationship between false-positive rate and precision depends on the class ratio. 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 detections. By contrast, at a precision of 90%, only 10% of all alarms would be false alarms.

5 Network-flow analysis

This section presents our architecture that jointly detects infected clients and malicious domains, as well as reference models that we will compare against.

5.1 Sluice network

Figure 1 shows the *sluice* network architecture for joint flow classification with soft parameter sharing. The left-hand part of the network classifies flows by whether they originate from infected clients, the right-hand part classifies flows by whether they are between a client and a malicious domain. The input features are derived from a window of $2k+1$ flows for a given client that is centered around the flow to be classified. The first stage of the network—the

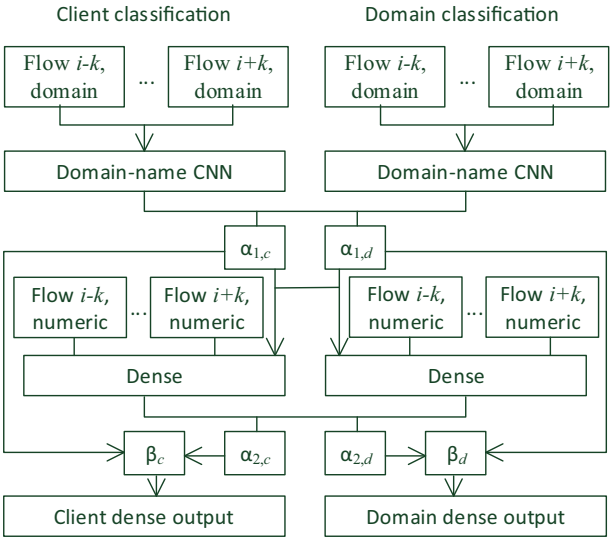


Fig. 1 Sluice dense on domain CNN

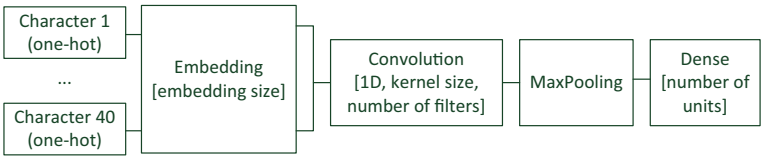


Fig. 2 Domain CNN architecture

domain-name CNNs—receives the domain names of the host domains within that window as input.

Figure 2 shows this domain-name CNN in more detail. It has a standard convolutional architecture with convolutional, max-pooling, and dense layers. Domain names are first represented as one-hot-encoded character sequences of the up to 40 last characters of a domain name. We selected the value of 40 because further increasing this parameter does not change the experimental results. In the next step, an embedding layer reduces this dimensionality; weights are shared for the embedding of each character. This is followed by a one-dimensional convolutional layer, a max-pooling layer, and a dense layer that constitutes the final encoding of the domain name.

The following dense layers receive the window of $2k + 1$ domain-name embeddings. Additionally, they receive a vector of numeric features for each of the $2k+1$ flows in the input window. The numeric attributes consist of the log-transformed duration, log-transformed numbers of sent and received bytes, duration, and the time gap from the preceding flow. These dense layers are followed by softmax output layers.

After each stage, the output from either side of the network is combined into a weighted average controlled by coupling coefficients α. Values of $\alpha_{\cdot,\cdot} = 0$ correspond to independent networks. In addition, the output layer is allowed to draw on all intermediate layers. The output of each hidden layer is weighted by a coefficient β and all weighted outputs are concatenated. Setting all the $\beta_{\cdot}$ values associated with the first hidden layer to zero and all

values associated with the second hidden layer to one correspond to the standard layered feed-forward architecture. We use the ReLU activation function for hidden layers.

The model is trained by using backpropagation on labeled network flows. At application time, detection results at the level of clients and domains are derived by maximizing the output scores of the positive class "infected client" over all network flows between the given client and any domain over an interval of 24 hours. A client is flagged as soon at this maximum exceeds a threshold value. Likewise, the output scores of the positive class "malicious domain" on the right-hand side is maximized over all flows between any client and the domain to be classified for 24 hours.

5.2 Independent models and hard sharing

Separating the left- and right-hand side of the sluice architecture constitutes the first natural baseline. We will refer to these models as *independent* models. This is equivalent to setting $\alpha_{\cdot,\cdot} = 0$, setting all the $\beta_{\cdot}$ values associated with the first hidden layer to zero and all values associated with the second hidden layer to one. The next natural baseline is *hard parameter sharing*. Here, only the output layers of the client- and domain-classification models have independent parameters while the domain CNN and the following dense layer exist only once.

5.3 LSTM on word2vec

This baseline model (Prasse et al. 2017) uses the word2vec continuous bag-of-words model (Mikolov et al. 2013) to embed domain names, and processes the flow sequence with an LSTM. 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. The input layer is fully connected to a hidden layer that implements the embedding. 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.

We use the weight matrix and configuration of Prasse et al. (2017) and refer to this model as *LSTM on word2vec*. This model uses character 2-grams, resulting in 1583 character 2-grams. Prasse et al. (2017) have found the *LSTM on word2vec* model to outperform a random-forest model. We therefore consider *LSTM on word2vec* to be our reference and do not include random forests in our experiments.

5.4 Metric space learning

Lokoč et al. (2016) extract a vector of soft histogram features for the flows between any client and a given domain. They apply a k-NN classifier in order to identify domains that are contacted by malware. We apply this approach to our problem of detecting malicious domains. We use exact inference instead of the approximate inference proposed by Lokoč et al. (2016). We prop this baseline up by additionally providing a list of engineered domain features described by Franc et al. (2015) as input to the classifier; we refer to this method as *4-NN soft histograms*.

Table 5 Best hyperparameters found using hyperband for models with shared blocks

Sluice		Hard parameter sharing	
Hyperparameter	Value	Hyperparameter	Value
Domain CNN			
Embedding size	128	Embedding size	64
Kernel size	16	Kernel size	16
Filters	512	Filters	512
Dense units	256	Dense units	64
Flow class.			
Dense units	512	Dense units	512
Window size	11	Window size	7

Table 6 Best hyperparameters found using hyperband for independent models

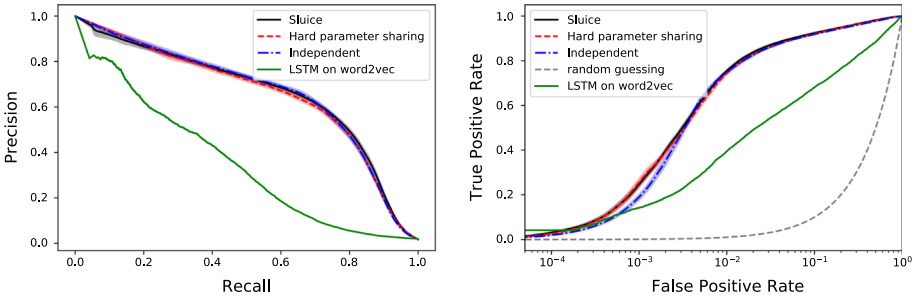
Independent client		Independent domain	
Hyperparameter	Value	Hyperparameter	Value
Domain CNN			
Embedding size	64	Embedding size	64
Kernel size	16	Kernel size	16
Filters	128	Filters	128
Dense units	32	Dense units	32
Flow class.			
Dense units	512	Dense units	512
Window size	9	Window size	9

6 Experiments

This section reports on malware-detection and malicious-domain-detection accuracy. We train all models on a single server with 40-core Intel(R) Xeon(R) CPU E5-2640 processor and 128 GB of memory. We train all neural networks using the Keras (Chollet et al. 2015) and Tensorflow (Abadi et al. 2015) libraries on a GeForce GTX TITAN X GPU using the NVidia CUDA platform. We implement the evaluation framework using the scikit-learn (Pedregosa et al. 2011) machine learning package.

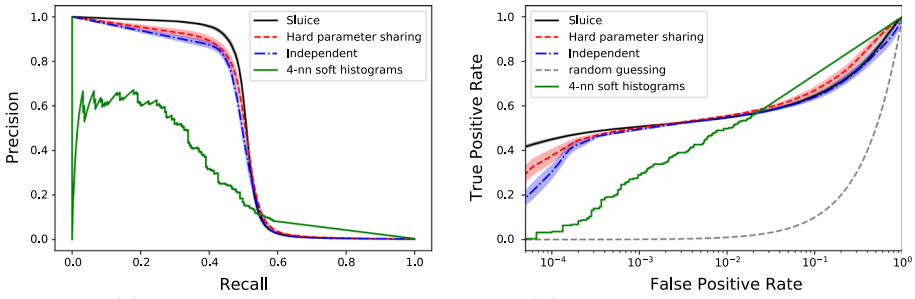
6.1 Parameter optimization

We optimize the hyperparameters of all networks on the training data using the hyperband algorithm (Li et al. 2016). For the domain CNN, we vary the embedding size between 2^5 and 2^7, the kernel size between 2 and 2^4, the number of filters between 2 and 2^9, and the number of dense units between 2^5 and 2^9. For the *sluice network*, the *independent models* and *hard parameter sharing*, we vary the number of dense units between 2^5 and 2^{11}, and the window size between 1 and 15 flows. Tables 5 and 6 shows the hyperparameter values after optimization.



(a) Precision-recall curves **(b)** ROC curves (log-scale for FPR)

Fig. 3 Detection of infected clients



(a) Precision-recall curves **(b)** ROC curves (log-scale for FPR)

Fig. 4 Detection of malicious domains

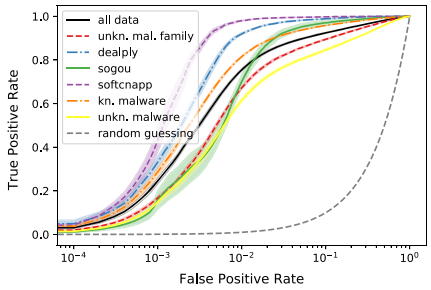
6.2 Infected clients: performance

We train the models on the *training data* and evaluate them on *test data* that was recorded after the training data. Figure 3 compares precision-recall and ROC curves; curves are averaged over 10 random restarts with Glorot initialization, colored bands visualize plus and minus one standard error.

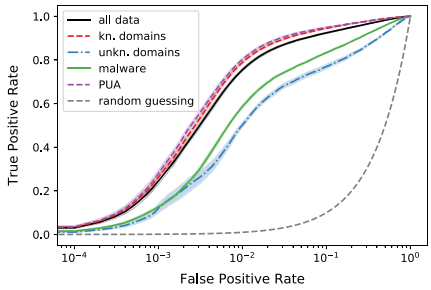
For malware detection, the *sluice network*, the *independent models* and *hard parameter sharing* differ only marginally in performance. All three detect 40% of malware with a precision of 80%. Based on Welch's t-test with significance level $\alpha = 0.05$, at false-positive rates of 10^{-4} and 10^{-3}, the *sluice network* is still significantly better than the *independent model* ($p = 0.021$ for 10^{-4} and $p = 0.008$ for 10^{-3}), but the difference between *sluice* and *hard parameter sharing* is not significant. *LSTM over word2vec* clearly performs substantially worse.

6.3 Malicious domains: performance

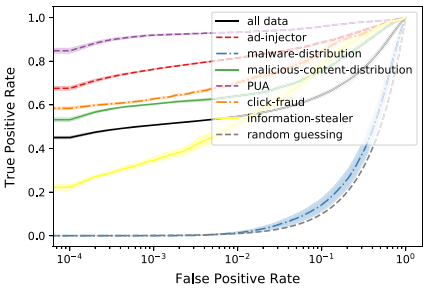
Figure 4 compares the model's performance for detection of malicious domains. Here, the precision-recall and ROC curves of the *sluice network* look favorable compared to the baselines. Intuitively, since there are fewer malicious domains in the training data than there are infected clients, it is plausible that malicious-domain detection benefits more strongly from transfer learning. The *4-NN soft histogram* baseline performs substantially worse. The



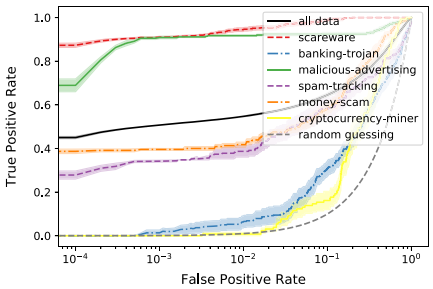
(a) Infected clients: different malware families.



(b) Infected clients: types of malware.



(c) Malicious domains: subgroups of domains.



(d) Malicious domains: subgroups of domains.

Fig. 5 *Sluice network* on subgroups of instances, ROC curves for infected clients (left figure) and malicious domains (right figures)

precision-recall curve becomes noisy near a recall of zero because for a low recall, the precision estimate is based on a small number of positives, and the decision function assumes a high value for several true and false positives.

Based on Welch's t-test with significance level $\alpha = 0.05$, at a false-positive rate of 10^{-4}, the *sluice network* performs significantly better than both the *independent model* ($p = 0.028$) and *hard parameter sharing*. At 10^{-3}, the *sluice network* outperforms the *independent model* ($p = 0.001$); it detects 40% of all malicious domains almost without false alarms. The difference between *sluice* and *hard parameter sharing* is not significant.

6.4 Detailed analysis

In this section, we study how the detection models perform on specific subgroups of clients and domains. We train a single model on all training data. In order to determine the performance for specific types of instances, we skip all other positive instances in the evaluation data. Since the class ratios vary widely between subgroups, we compare ROC curves. Figure 5a shows that the most popular malware families can be detected more easily, which corresponds to their high prevalence in the training data. Perhaps surprisingly, we see that the model detects unknown variations of known malware families as well as *unknown malware families*—that is, malware families for which no representative is present in the training data—almost as accurately as known malware. Figure 5b shows that the model's performance is just slightly better for the highly prevalent potentially unwanted applications ("PUA") than

it is for malware. We also see that malware which does not contact any domain that occurs in the training data (labeled "unknown domains") is detected with comparable accuracy to malware that contacts known domains.

Figure 5c, d show how the *sluice network* performs on specific types of malicious domains. Here, we see that the detection performance uniformly depends on the prevalence of the domain type in the training data. Only 10 backends for banking trojans are included in the training data, and no single cryptocurrency-mining backend. Malware-distribution servers are almost impossible for the model to detect, despite being the second-most frequent type of malicious domains in the training data. But a detailed analysis shows that the training data contains only 1447 flows (out of more than 44 million) from malware-distribution servers; so at the level of flows, this class is actually rare.

6.5 Additional experiments

We carry out additional experiments but omit the detailed results for brevity. The *independent models* differ from *LSTM on word2vec* in two aspects: the use of the domain-name CNN instead of a word2vec embedding, and the choice of processing windows of network flows in a convolutional way with max-pooling over all window positions instead of an LSTM. In order to explore whether performance differences are due to the different domain-name embedding or the different handling of the time-series input as additionally experiment with the intermediate form of *dense on word2vec*. We find that while this architecture performs significantly better than *LSTM on word2vec*, it still performs much worse than *independent models*.

The *4-NN soft histogram model* (Lokoč et al. 2016) originally does not use the engineered domain features (Franc et al. 2015) that we provide it with. We find that using the *4-NN* model without the domain features (or using the domain features without the histogram features) deteriorates the results. We also find that combining the soft-histogram features and engineered domain features with a random forest improves the result over the 4-NN classifier, but its performance remains substantially below the performance of all neural networks. Finally, we find that adding additional convolutional and max-pooling layers or replacing the dense layer in the sluice network with convolutional and max-pooling layers deteriorates its performance.

7 Conclusion

Detection of malware-infected clients and malicious domains allows organizations to use a centralized security solution that establishes a uniform security level across the organization with minimal administrative overhead. A specifically prepared VPN client makes it possible to collect large amounts of HTTPS network traffic and label network flows in retrospect by whether they originate from malware. This makes it possible to employ relatively high-capacity prediction models. By contrast, malicious domains have to be identified by means of an expensive forensic analysis. We have developed a method that jointly detects infected clients and malicious domains from encrypted network traffic without compromising the encryption.

We can draw a number of conclusions. All network architectures that we study improve on the previous state of the art by a large margin. We find that transfer learning using a sluice network improves malware-detection—for which we have a large body of training data—

slightly over learning independent networks. Transfer learning allows us to leverage the large body of malware training data to improve the detection of malicious domains. The sluice network detects 40% of all malware with a precision of 80% using only encrypted HTTPS network traffic—at this threshold level, 20% of all alarms are false alarms. In practice, each alarm triggers the notification of a security analyst; if 80% of the notifications indicate an actual security breach, an analyst will not get the impression that the notification system can be ignored. The sluice network detects new variants of known malware families and malware of families that have not yet been known at training time with nearly the same accuracy. This finding is remarkable because signature-based antivirus tools cannot detect such malware. The network also detects 40% of all malicious domains with a precision of nearly 1. Given the high costs of a manual analysis of domains, this result has a potentially high impact for network security in practice.

Acknowledgements The work of Tomáš Pevný has been partially funded by Czech Ministry of education under the GACR project 18-21409S. We would like to thank Virustotal.com for their kind support.

Funding Funding was provided by Cisco R&D.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from https://www.tensorflow.org/. Accessed 6 Sept 2018.

Anderson, B., Quist, D., Neil, J., Storlie, C., & Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal of Computer Virology*, *7*(4), 247–258.

Argyriou, A., Evgeniou, T., & Pontil, M. (2007). Multi-task feature learning. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems 19* (pp. 41–48). MIT Press.

Arora, A., Garg, S., & Peddoju, S. K. (2014). Malware detection using network traffic analysis in android based mobile devices. In *International conference on next generation mobile apps, services and technologies* (pp. 66–71).

Bartos, K., & Sofka, M. (2015). Robust representation for domain adaptation in network security. In *European conference on machine learning and principles and practice of knowledge discovery in databases* (pp. 116–132). Springer.

Bartos, K., Sofka, M., & Franc, V. (2016). Optimized invariant representation of network traffic for detecting unseen malware variants. In *USENIX security symposium* (pp. 807–822).

Baxter, J. (1997). A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine Learning*, *28*(1), 7–39.

Bickel, S., Bogojeska, J., Lengauer, T., & Scheffer, T. (2008). Multi-task learning for hiv therapy screening. In *Proceedings of the international conference on machine learning* (pp. 56–63). ACM.

Blum, S. B., & Lueker, J. (2001). Transparent proxy server, January 30. US Patent 6,182,141.

Caruana, R. (1993) Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the international conference on machine learning*.

Chollet, F., et al. (2015). Keras. https://keras.io. Accessed 6 Sept 2018.

Crotti, M., Dusi, M., Gringoli, F., & Salgarelli, L. (2007). Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, *37*(1), 5–1.

Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., et al. (2018). Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*. https://doi.org/10.1109/TDSC.2017.2700270.

Duong, L., Cohn, T., Bird, S., & Cook, P. (2015). A neural network model for low-resource universal dependency parsing. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 339–348).

Dusi, M., Crotti, M., Gringoli, F., & Salgarelli, L. (2009). Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, *53*(1), 81–97.

Evgeniou, T., Micchelli, C. A., & Pontil, M. (2005). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, *6*(Apr), 615–637.

Finkel, J. R., & Manning, C. D. (2009). Hierarchical bayesian domain adaptation. In *Proceedings of ACL human language technologies* (pp. 602–610).

Finley, K. (2017). Half the web is now encrypted. That makes everyone safer. *Wired*. https://www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer/.

Franc, V., Sofka, M., & Bartos, K. (2015). Learning detector of malicious network traffic from weak labels. In A. Bifet, M. May, B. Zadrozny, R. Gavalda, D. Pedreschi, F. Bonchi, J. Cardoso, & M. Spiliopoulou (Eds.), *Machine learning and knowledge discovery in databases* (pp. 85–99). Springer.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., et al. (2016). Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, *17*(59), 1–35.

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. arXiv:1705.03122.

Karim, M. E., Walenstein, A., Lakhotia, A., & Laxmi, P. (2005). Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, *1*(1–2), 13–23.

Kogan, R. (2015). Bedep trojan malware spread by the angler exploit kit gets political. Spider Labs Blog. https://www.trustwave.com/Resources/SpiderLabs-Blog/Bedep-trojan-malware-spread-by-the-Angler-exploit-kit-gets-political/. Accessed 6 Sept 2018.

Kohout, J., & Pevny, T. (2015a) Automatic discovery of web servers hosting similar applications. In *Proceedings of the IFIP/IEEE international symposium on integrated network management*.

Kohout, J., & Pevny, T. (2015b). Unsupervised detection of malware in persistent web traffic. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing*.

Lashkari, A., Kadir, A., Gonzalez, H., Mbah, K., & Ghorbani, A. (2015). Towards a network-based framework for android malware detection and characterization. In *Proceedings international conference on privacy, security, and trust*.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*. arXiv:1603.06560.

Lokoč, J., Kohout, J., Čech, P., Skopal, T., & Pevnỳ, T. (2016). k-NN classification of malware in HTTPS traffic using the metric space approach. In M. Chau, G. A. Wang, & H. Chen (Eds.), *Intelligence and security informatics* (pp. 131–145). Springer.

Long, M., & Wang, J. (2015). Learning multiple tasks with deep relationship networks. In arXiv:1506.02117.

Malik, J., & Kaushal, R. (2016). CREDROID: Android malware detection by network traffic analysis. In *Proceedings of the first ACM workshop on privacy-aware mobile computing* (pp. 28–36). ACM.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 26* (pp. 3111–3119). Curran Associates, Inc.

Misra, I., Shrivastava, A., Gupta, A., & Hebert, M. (2016). Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3994–4003).

Nelms, T., Perdisci, R., & Ahamad, M. (2013). Execscent: Mining for new C&C domains in live networks with adaptive control protocol templates. In *Proceedings of the USENIX security symposium*.

Nguyen, T., & Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys, Tutorials*, *10*(4), 56–76.

Pascanu, R., Stokes, J. W., Sanossian, H., Marinescu, M., & Thomas, A. (2015). Malware classification with recurrent networks. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing* (pp. 1916–1920). IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pevny, T., & Somol, P. (2016). Discriminative models for multi-instance problems with tree structure. In *Proceedings of the international workshop on artificial intelligence for computer security*.

Prasse, P., Machlica, L., Pevný, T., Havelka, J., & Scheffer, T. (2017). Malware detection by analysing network traffic with neural networks. In *Proceedings of the European conference on machine learning*.

Ruder, S., Bingel, J., Augenstein, I., & Søgaard, A. (2017). Sluice networks: Learning what to share between loosely related tasks. arXiv:1705.08142v1 [stat.ML]

Swinnen, A., & Mesbahi, A. (2014). One packer to rule them all: Empirical identification, comparison and circumvention of current antivirus detection techniques. *BlackHat USA*. https://www.blackhat.com/docs/us-14/materials/us-14-Mesbahi-One-Packer-To-Rule-Them-All-WP.pdf.

Wright, C. V., Monrose, F., & Masson, G. M. (2006). On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, *7*, 2745–2769.
Yang, Y., & Hospedales, T. M. (2016). Trace norm regularised deep multi-task learning. arXiv:1606.04038.