Lightweight Behavioral Malware Detection for Windows Platforms

Bander Alsulami
Drexel University
bma48@drexel.edu

Avinash Srinivasan
Temple University
avinash@temple.edu

Hunter Dong
Temple University
hunter@temple.edu

Spiros Mancoridis
Drexel University
spiros@drexel.edu

Abstract

We describe a lightweight behavioral malware detection technique that leverages Microsoft Windows prefetch files. We demonstrate that our malware detection achieves a high detection rate with a low false-positive rate of 1×10^{-3}, and scales linearly for training samples. We demonstrate the generalization of our malware detection on two different Windows platforms with a different set of applications. We study the loss in performance of our malware detection in case of concept drift and its ability to adapt. Finally, we measure our malware detection against evasive malware and present an effective auxiliary defensive technique against such attacks.

1 Introduction

Static signature-based malware detection methods use static properties of programs to discriminate between benign and malicious programs. Static signature-based malware detection requires examining the malware to create a distinct signature for each newly discovered malware. A signature can be based on a byte-code sequence [14], binary assembly instruction [18], an imported Dynamic Link Library (DLL) [23], or function and system calls [22, 1]. Unfortunately, malware authors use various obfuscation techniques to generate new variants of the same malware [27]. Therefore, the number of signatures grows rapidly as well as the time it takes to analyze and create each signature. This endangers critical systems and increases the spread of malware infection.

In this paper, the concept of prefetching takes a slight detour from the conventional realm. The concept is only evidenced on platforms running the Windows operating system starting with Windows XP. The objective of prefetching is to make resources available to the processor prior to an explicit request. This involves analyzing and predicting the behavior of programs running on Windows platforms. Prefetch files have drawn attention from the computer forensics community and law enforcement agencies. However, no prior work in malware detection has investigated the usage of prefetch files as dynamic features for behavioral malware detection.

2 Windows Prefetch Background

Prefetch files date back to the Windows XP operating system. Prefetching was introduced to speed up the booting process and launch time of applications. Prefetching has also been extended in Windows Vista by *SuperFetch* [21]. SuperFetch attempts to accelerate application launch times by monitoring and adapting to applications' usage patterns over time. SuperFetch caches the majority of the files and data needed by the applications in advance, so that they can be accessible quickly later. The prefetching process occurs when the *Windows Cache Manager* (WCM) monitors certain elements of data that are extracted from the disk into memory by processes. This monitoring occurs during the first two minutes of the booting process, and for another sixty seconds after all the system services are loaded. Similarly, after an application is executed, the WCM monitors the first ten seconds. WCM stores dependency files for each application in files with *.PF* extensions inside a system folder called *Prefetch*. For instance, when a user executes an application such as *Notepad*, the system generates the application prefetch file name and looks in the prefetch folder for a match. If the lookup results in a match, the WCM notifies the operating system to read the *Notepad* prefetch file and open directories or files referenced in that prefetch file. Otherwise, a new prefetch file is created for that application.

3 Related Work

Behavioral malware detections have been proposed to overcome the limitations of the static signature-based

978-1-5386-1436-5/17/$31.00 ©2017 IEEE

malware detection [10]. Such detection captures the run-time behavior of malware during its execution. Behavioral malware detection relies on various dynamic properties as features such as a file system activities [26], terminal commands [28], network communications [30], and system calls [31, 12]. The detection function can be designed as in rule-based or learned such as in machine learning algorithms. Machine learning techniques have been researched extensively to build intelligent models that discriminate between the benign and malicious processes.

Behavioral malware detectors on dynamic systems become inconsistent and ineffective over time [13]. The work in [15] proposes an online learning technique to continuously updating the learning model. The work in [4] proposes paired detectors to automatically respond to changes in systems. The detectors use global and local incidents to create a stable performance over the course of execution. Drifting is not only limited to changes in the benign processes, as malware families also exhibit evolution in the behavior over time to avoid detection. The work in [24] evaluates the drifting in three malware families and n-gram detection models. The study proposes a general method to track drift in malware families. The study shows that a negligible drift in behavioral malware detection can be exploited by malware to avoid detection.

4 Malware Detection Framework

Our malware detector discriminates between normal and malicious Windows applications using prefetch files found in the Windows Prefetch folder. We use machine learning techniques to implement the components of our detector. This section describes the five major components of the malware detector: Feature Extraction, Feature Scaling and Transformation, Dimensionality Reduction, and Detection Classifier.

4.1 Feature Extraction and Transformation

Our malware detector uses a Bag of Words (BoW) model to represent the list of dependency file names in a prefetch file. BoW models are used extensively in document classification and natural language processing [11]. Each document is represented by a vector of the frequencies of all words occurring in the document. In our case, each trace is viewed as a sequence of n-grams. An n-gram is a sequence of n adjacent dependecy file names [7]. After the feature vectors are extracted, A *Term Frequency-Inverse Document Frequency* (TF-IDF) transformation is applied. TF-IDF is a technique that highlights important n-grams in feature vectors [20]. Rare n-grams receive a higher weight than the common n-grams across all feature vectors. Although TF-IDF is traditionally used in information retrieval and text mining, the technique has shown success in intrusion detection [17].

4.2 Dimensionality Reduction

N-gram models have a high-dimensional feature space, which may be computationally intractable. This is known as the curse of dimensionality [16]. Dimensionality reduction techniques can be used to transform the data into a useful lower dimensional representation while retaining a high variance of the original data. While reduction techniques can be linear or non-linear, the linear reduction techniques provide stable results over a high number of dimensions [5]. Thus, our malware detector uses a linear dimensionality reduction technique called Singular Value Decomposition [9].

4.3 Randomized Feature Selection

Our malware detector uses a simple randomization feature selection technique to increase its resilience without affecting its detection accuracy and run-time performance. The technique, called *Dropout*, is based on randomly removing features during the training process [25]. Thus, the malware detector does not relay on a small set of features that might be exploited by a malicious program.

4.4 Detection Classifier

Malware detection can be defined as a binary classification problem. That is, the training data is sampled from two classes: the benign and malicious classes. Therefore, we use a Logistic Regression (LR) classifier for class prediction. LR is suitable for machine learning problems with binary classes. LR is a Generalized Linear Regression (GLM) with a non-linear function called sigmoid, also known as the logistic function.

5 Experimental Setup

This section describes our experimental setup, the collected datasets, and the ground truth labeling used to evaluate our malware detector.

5.1 Dataset Collection

To evaluate our malware detector, we conduct an experiment on two different Windows platforms. Each platform generates a separate dataset that includes prefetch files samples for benign and malware programs. In the first dataset, we collect the prefetch files from the Windows 7 platform and name the dataset as Prefetch-7. The second dataset is named Prefetch-10 and includes prefetch files from Windows 10.

5.2 Ground Truth

Ground truth labels for malware are obtained through an online third-party virus scanning service. In this experiment, we use VirusTotal[1]. Given an MD5, SHA1 or SHA256 of a malware file, VirusTotal provides the detection information for popular anti-virus engines. This information also includes target platforms, malware types, and malware families. We exclude any malware that are not identified by any anti-virus detection engine.

6 Malware Framework Evaluation

In this section we evaluate the major components of the malware detection framework.

6.1 Detection Performance

To show the effectiveness of our malware detector on the prefetch datasets, we compare our LR detectors to Support Vector Machine (SVM) detectors [2]. SVM have established state-of-the-art results in multiple malware detection and classification research [32, 19, 6]. We compare our [2,3]-Grams LR detectors to the best SVM detectors from [6]. We use 10-fold cross-validation with stratified sampling to create a balanced distribution of benign and malware samples in each fold.

Table 2 shows the TPR, FPR, and AUC metrics for [2,3]-Grams LR and [3,4]-Grams SVM detectors. On the Prefetch-7 dataset, [2,3]-Grams LR detectors achieve as high as 0.997 TPR on 1.2×10^{-3} FPR. On the contrary, SVM detectors achieve a lower TPR at a higher FPR. On the Prefetch-10 dataset, [2,3]-Grams LR detectors achieve 1.0 TPR and zero to 8.4×10^{-5} FPR, which is the ideal FPR for practical malware detection [3]. This experiment shows that LR detectors are superior to SVM detectors on prefetch datasets.

6.2 Run-time Performance

Figure 1 shows run-time performance for LR [2,3]-Grams LR, and [3,4]-Grams SVM detectors during the training and evaluation processes. The run-time of SVM detectors grow quadraticically with the number of the traces. On the contrary, LR detectors maintain a linear growth on a large number of traces. Therefore, LR detectors, trained using Stochastic Gradient Descent (SGD), are scalable and practical to a large number of traces. It is worth mentioning that SVM can also be trained using SGD. SGD works by minimizing the error given by the hinge loss function. The hinge loss function is a maximum margin classification function [2]. Therefore, the run-time performance for SVM, trained on SGD, should be similar to LR detector since they are using the same optimization algorithm for training.

6.3 Drift Evaluation

Malware detectors that do not adapt adequately to the new changes in the platforms become inconsistent and ineffective over time. The change in programs behavior over time, called *Concept Drift*, is a well-known issue in malware and intrusion detection on dynamic systems [8]. Concept drift is not only limited to changes in the benign programs, malware families also evolve their behavior over time to avoid detection [24].

Tables 3 shows the TPR and FPR when the detector is trained on Windows 7 platform and evaluated on Windows 10 platform. The loss in TPR across the dataset is less than 1×10^{-2}. Moreover, the increase in FPR is no more than 1×10^{-2}. The loss in performance can be accounted because of the different configuration sets in the Windows platforms, newly installed programs, and different cache entries in the Windows Prefetch folder since the folder is updated regularly. Therefore, the loss in performance is very minimum when the platform operating system is upgraded a new version.

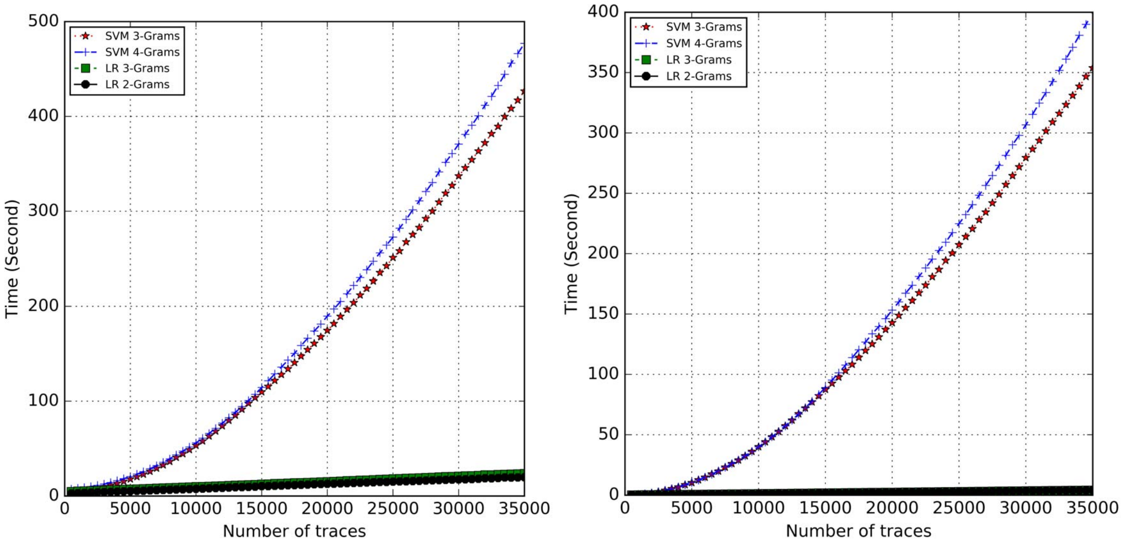
While training our malware detectors from scratch is fast, we also study incremental training to measure the training time required to improve the detector when the detection accuracy drops. We compare the detection accuracy of the incrementally trained detector to the newly trained detector. Figure 2 shows the TPR over the course of the training process. We select four different versions of our malware detector: new [2,3]-Grams LR , and trained [2,3]-Grams LR. The trained detectors are pre-trained on the Prefetch-7 dataset. The Prefetch-7 dataset represents the old state of the platform, while the Prefetch-10 dataset represents the new state. We train the detectors on Prefetch-10 to represent the new changes in the platform. We record the TPR and FPR after each epoch until 100 epochs.

The experiment shows that the retrained detectors achieve a higher TPR and a lower FPR more quickly than the newly trained detectors. In fact, the training process can be stopped early for retrained detectors, which is an advantage. Therefore, incremental training for our malware detector is efficient and recommended over fully retraining.

[1] VirusTotal, http://www.virustotal.com

Malware Type	Size	Malware Family
Adware	573	EoRezo, Firseria, IBryte, MultiPlug, Nsis, Kranet, PullUpdate
Backdoor	10	Advml, Fynloski, Cycbot, BackDoor2
Trojan	690	TrojanDownloader, TrojanDropper, InstalleRex, Kazy, Muldrop, Swizzor, TrojWare, Hlux, VBInj, Kryptik
Virus	55	Sality, Bandoo, Smshoax, Oxypumper, DomaIQ, MalSign
Worm	28	Downadup, WormDropper, Korgo

Table 1: Malware types, each type size, and samples of malware families according to Eset-Node32, kaspersky, and Symantec engines



(a) Training Time

(b) Evaluation Time

Figure 1: The training and evaluation run-times for malware detectors on the Prefetch-10 dataset with random over-sampling.

		Prefetch-7		
Classifier	BoW	TPR	FPR	AUC
SVM	3	0.97	5.2×10^{-2}	0.991
	4	0.97	5.7×10^{-2}	0.985
LR	2	0.997	1.2×10^{-3}	0.999
	3	0.997	1.2×10^{-3}	0.999
		Prefetch-10		
Classifier	BoW	TPR	FPR	AUC
SVM	3	0.996	1.09×10^{-3}	0.998
	4	0.995	2.35×10^{-3}	0.998
LR	2	1.0	0.0	1.0
	3	1.0	8.4×10^{-5}	1.0

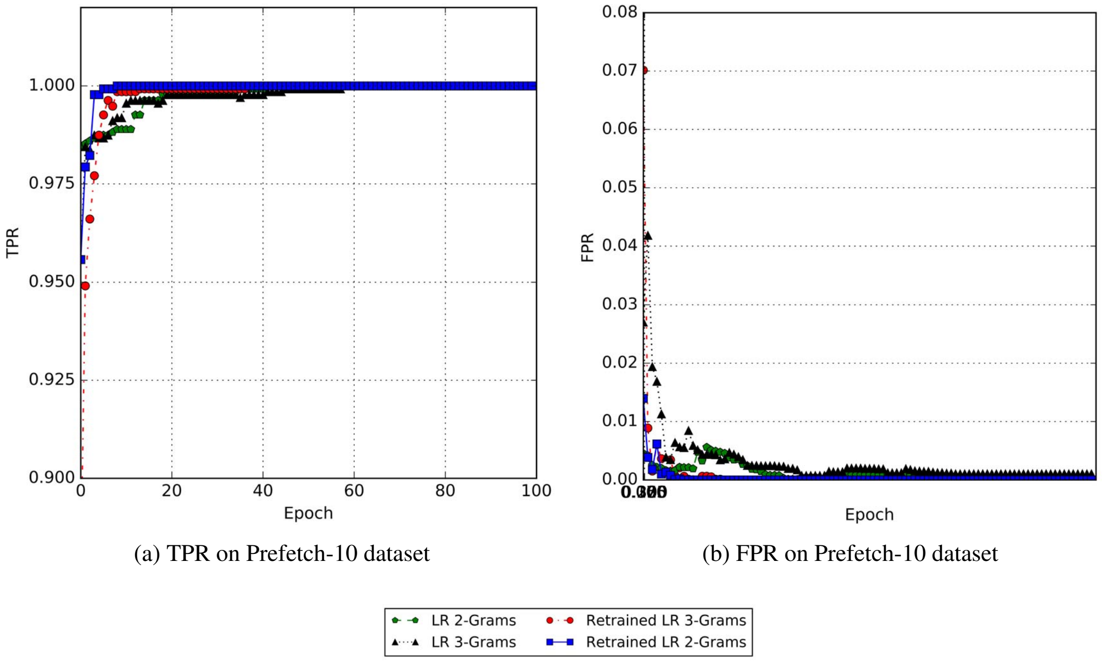
Table 2: TPR,FPR, and AUC for [2,3]-Grams LR and [3,4] SVM detectors.

Detector	TPR	FPR
2-Grams LR	0.9919**(-0.008)**	0.0074**(+0.007)**
3-Grams LR	0.9911**(-0.009)**	0.0062**(+0.006)**

Table 3: The loss in TPR and FPR when the detector is evaluated on a different Windows platform than the platform it was trained on.

7 Detecting Malware Evasion

To evaluate the effectiveness of our randomized feature selection technique, we implement a general method to generate evasive malware from our samples [29]. The method appends benign traces to malware traces to evade malware detection. While the method uses genetic programming to find the right mutation to succeed, we re-



(a) TPR on Prefetch-10 dataset

(b) FPR on Prefetch-10 dataset

Figure 2: The TPR and FPR for newly trained and retrained [2,3]-Grams LR detectors.

place genetic programming with a simple appending operation. In each iteration, we append a benign trace to all the malware traces and measure the decrease in detection accuracy. Next, we increase the length of the benign trace and repeat the process until the end of the benign trace. We apply the same process for a randomly selected subset of benign traces and average the detection scores across them.

Figure 3 shows the effect of applying Dropout during the training process. We use the F1 score because it incorporates True Positive, False Positive, and False Negative metrics. We train our [2,3]-Grams malware detectors with and without dropout. The figure shows that malware detectors trained with dropout are more robust to evasive malware. [2,3]-Grams LR detectors trained with Dropout decrease their detection accuracy more slowly despite adding longer traces of file names from benign prefetch files. We also find that higher order n-grams are more resilient than lower order n-grams. 3-Grams LR detectors are less affected by evasive traces than 2-Grams LR detectors. This experiment shows that a simple randomization technique can help improve the robustness of the malware detector without increasing the problem's complexity or decreasing the run-time performance of the detector.

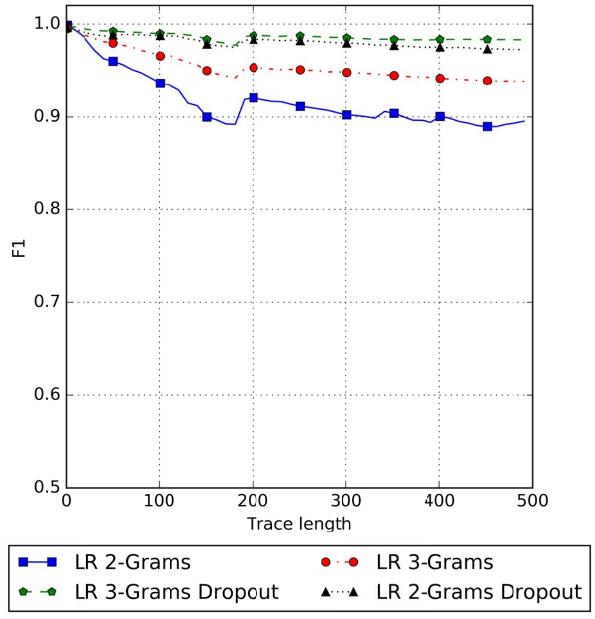


Figure 3: F1 for [2,3]-Grams LR detectors trained with and without Dropout.

8 Conclusions and Future Work

We demonstrate that our malware detector is able to adapt to new information and changes in the environment without decreasing its accuracy or increasing its performance overhead. We also studied the resilience of our malware detector to malware evasive techniques. This study led us to create a simple randomization solution to harden the malware detector.

In the future, we would like to include techniques to detect when the detection accuracy degrades and adapt accordingly. Moreover, we would like to further test our malware detector against increasingly sophisticated evasive malware.

9 Acknowledgments

This work is supported by a Fellowship from the Isaac L. Auerbach Cybersecurity Institute at Drexel University. We thank Alexander M. Duff for assistance with the reviewing and for the comments that greatly improved the readability of manuscript.

References

[1] Alazab, M., Venkatraman, S., Watters, P., and Alazab, M. Zero-day malware detection based on supervised learning algorithms of api call signatures. In *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121* (2011), Australian Computer Society, Inc., pp. 171–182.

[2] Anzai, Y. *Pattern recognition and machine learning*. Elsevier, 2012.

[3] Axelsson, S. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security* (1999), ACM, pp. 1–7.

[4] Bach, S. H., and Maloof, M. A. Paired learners for concept drift. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (2008), IEEE, pp. 23–32.

[5] Banda, J. M., Angryk, R. A., and Martens, P. C. Quantitative comparison of linear and non-linear dimensionality reduction techniques for solar image archives. In *FLAIRS Conference* (2012).

[6] Canzanese, R., Mancoridis, S., and Kam, M. System call-based detection of malicious processes. In *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on* (2015), IEEE, pp. 119–124.

[7] Cavnar, W. Using an n-gram-based document representation with a vector processing retrieval model. *NIST SPECIAL PUBLICATION SP* (1995), 269–269.

[8] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR) 46*, 4 (2014), 44.

[9] Golub, G. H., and Reinsch, C. Singular value decomposition and least squares solutions. *Numerische mathematik 14*, 5 (1970), 403–420.

[10] Jacob, G., Debar, H., and Filiol, E. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology 4*, 3 (2008), 251–266.

[11] Joachims, T. *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers, 2002.

[12] Kang, D.-K., Fuller, D., and Honavar, V. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC* (2005), IEEE, pp. 118–125.

[13] Kantchelian, A., Afroz, S., Huang, L., Islam, A. C., Miller, B., Tschantz, M. C., Greenstadt, R., Joseph, A. D., and Tygar, J. Approaches to adversarial drift. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security* (2013), ACM, pp. 99–110.

[14] Kolter, J. Z., and Maloof, M. A. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research 7*, Dec (2006), 2721–2744.

[15] Lane, T., and Brodley, C. E. Approaches to online learning and concept drift for user identification in computer security. In *KDD* (1998), pp. 259–263.

[16] Langley, P. *Elements of machine learning*. Morgan Kaufmann, 1996.

[17] Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2014.

[18] Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., and Elovici, Y. Unknown malcode detection using opcode representation. In *Intelligence and Security Informatics*. Springer, 2008, pp. 204–215.

[19] Peiravian, N., and Zhu, X. Machine learning for android malware detection using permission and api calls. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on* (2013), IEEE, pp. 300–305.

[20] Rajaraman, A., Ullman, J. D., Ullman, J. D., and Ullman, J. D. *Mining of massive datasets*, vol. 1. Cambridge University Press Cambridge, 2012.

[21] Russinovich, M. Inside the windows vista kernel: Part 3. *Microsoft TechNet Magazine* (2007).

[22] Schmidt, A.-D., Bye, R., Schmidt, H.-G., Clausen, J., Kiraz, O., Yuksel, K. A., Camtepe, S. A., and Albayrak, S. Static analysis of executables for collaborative malware detection on android. In *Communications, 2009. ICC'09. IEEE International Conference on* (2009), IEEE, pp. 1–5.

[23] Schultz, M. G., Eskin, E., Zadok, F., and Stolfo, S. J. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on* (2001), IEEE, pp. 38–49.

[24] Singh, A., Walenstein, A., and Lakhotia, A. Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence* (2012), ACM, pp. 81–92.

[25] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[26] Stolfo, S. J., Apap, F., Eskin, E., Heller, K., Hershkop, S., Honig, A., and Svore, K. A comparative evaluation of two algorithms for windows registry anomaly detection. *Journal of Computer Security 13*, 4 (2005), 659–693.

[27] Toderici, A. H., and Stamp, M. Chi-squared distance and metamorphic virus detection. *Journal of Computer Virology and Hacking Techniques 9*, 1 (2013), 1–14.

[28] Wang, K., and Stolfo, S. One-class training for masquerade detection.

[29] Xu, W., Qi, Y., and Evans, D. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium* (2016).

[30] Ye, N., and Chen, Q. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International 17*, 2 (2001), 105–112.

[31] Ye, N., Li, X., Chen, Q., Emran, S. M., and Xu, M. Probabilistic techniques for intrusion detection based on computer audit data. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 31*, 4 (2001), 266–274.

[32] Ye, Y., Li, T., Zhu, S., Zhuang, W., Tas, E., Gupta, U., and Abdulhayoglu, M. Combining file content and file relations for cloud based malware detection. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (2011), ACM, pp. 222–230.