# Graph-based Malware Detection Using Dynamic Analysis

**Blake H. Anderson, Daniel A. Quist, Joshua C. Neil, ACS-PO; Curtis B. Storlie, CCS-6; Terran Lane, University of New Mexico, Michael E. Fisk, ACS-PO**

We introduce a novel malware detection algorithm based on the analysis of graphs that are constructed from dynamically collected instruction traces of the target executable. These graphs represent Markov chains, where the vertices are the instructions and the transition probabilities are estimated by the data contained in the trace. We use a combination of graph kernels to create a similarity matrix between the instruction trace graphs. The resulting graph kernel measures similarity between graphs on both local and global levels. Finally, the similarity matrix is sent to a support vector machine to perform classification. Our method is particularly appealing because we do not base our classifications on the raw n-gram data, but rather use our data representation to perform classification in graph space. Our results show a statistically significant improvement over signature-based and other machine-learning-based detection methods.

Malware continues to be an ongoing threat to modern computing. In 2010, more than 286 million unique variants of malware were detected [1]. Despite the majority of this new malware being created through polymorphism and simple code obfuscation techniques, and thus being very similar to known malware, it is still not detected by signature-based anti-virus programs [2]. With the ever-increasing proliferation of these threats, it is important to develop new techniques to detect and contain these malware.

Many of the current anti-virus programs available rely on a signature-based approach to classify programs as being either malicious or benign. Signature-based approaches are popular due to their low false positive rate and low computational complexity on the end host, both of which are appealing for daily usage. Unfortunately, these schemes have been shown to be easily defeated by simple code obfuscation techniques [2]. With the ease of creating a new virus through these techniques and polymorphic viruses becoming more prevalent, non-signature-based methods are becoming more attractive.

To combat these issues, several researchers began to look at less strict measures to detect malicious code. These methods have generally revolved around n-gram analysis of the static binary or dynamic trace of the malicious program [3]. These methods have shown great promise in detecting zero-day malware, but there are drawbacks related to these approaches. The two parameters generally associated with n-gram models are n, the length of the subsequences being analyzed, and L, the number of n-grams to analyze. For larger values of n and L, there is a much more expressive feature space that should be able to discriminate between malware and benign software more easily. But with these larger values of n and L, we run into the curse of dimensionality: the feature space becomes too large and we do not have enough data to sufficiently condition the model. With smaller values of n and L, the feature space becomes too small and discriminatory power is lost.

For our research, we use a modified version of the Ether Malware Analysis framework [4] to perform the data collection. Ether is a set of extensions on top of the Xen virtual machine. Malware frequently uses self-protection measures to thwart debugging and analysis. Ether uses a tactic of zero modification to be able to track and analyze a running system. Zero modifications preserve the sterility of the infected system, and reduce the methods that malware authors can use to detect if their malware is being analyzed. Increasing the complexity of detection makes for a much more robust analysis system. We use these modifications to allow for deeper introspection of the application programming interface (API) and import internals [5].

Our data representation gets away from the need to specify the appropriate n and L. Instead we model the dynamic instruction trace as a Markov chain represented by a weighted, directed graph. The instructions of the program are represented as vertices, and the weights of the edges are the transition probabilities of the Markov chain, which are estimated using the program trace we collect. An example

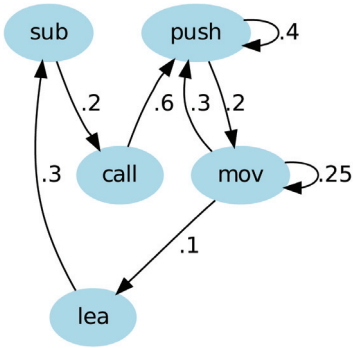


*Fig. 1. A hypothetical fragment of a full Markov chain graph. In a real Markov chain graph, all of the out-going edges would sum to 1.*

For more information contact Blake Anderson at banderson@lanl.gov

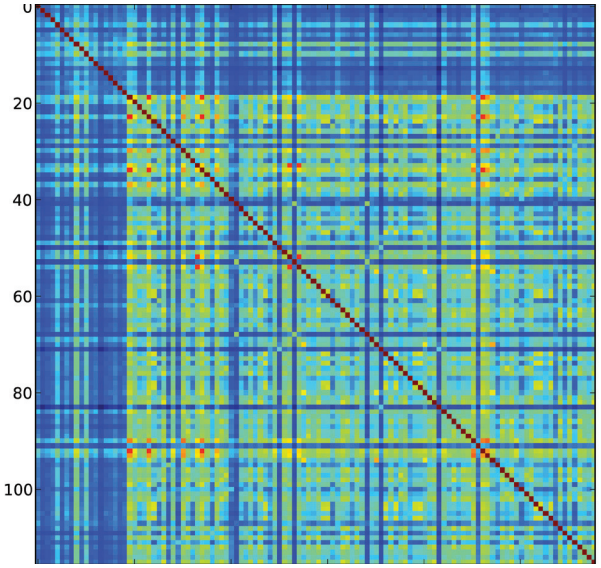


*Fig. 2. The heat maps of the kernel (similarity) matrix for benign software versus malware. The smaller block in the upper left of the figure represents benign software and the larger lower right block represents malware.*

of a hypothetical Markov chain graph is presented in Fig. 1. The novel contribution we present is to construct a similarity, or kernel, matrix between the Markov chain graphs and use this matrix to perform classification.

Our approach makes use of two types of kernels: a Gaussian kernel and a spectral kernel. The notions of similarity that these two kernels measure are quite distinct, and we found them to complement each other very well. The Gaussian kernel searches for local similarities between the adjacency matrices. It works by taking the exponential of the squared distances between corresponding edges in the weighted adjacency matrices. The motivation behind this kernel is that two different classes of programs should have different pathways of execution, which would result in a low similarity score.

The other kernel we use is based on spectral techniques [6]. These methods use the eigenvectors of the graph Laplacian to infer global properties about the graph. These eigenvectors encode global information about the graph's smoothness, diameter, number of components and stationary distribution, among other things. We then construct our second kernel by using a Gaussian kernel on these eigenvectors.

If we have two valid kernels, we are assured that a linear combination of these two kernels is also a valid kernel. This algebra on kernels allows us to elegantly combine kernels that measure very different aspects of the input data. We used cross-validation to find a suitable linear combination of our two kernels where the weights of the kernels are constrained to sum to one. Figure 2 shows a heat map of the combined kernel matrix with 19 instances of benign software and 97 instances of malicious software. Once the combined kernel matrix is constructed, we use a standard support vector machine to perform classification.

To validate our approach, we used a dataset composed of 615 instances of benign software and 1,615 instances of malware. We tested our combined kernel, Gaussian kernel, spectral kernel, and the n-gram methodology using a support vector machine, values of n ranging from two to six, and values of L ranging from 500 to 3,000 in increments of 500. We also tested against 10 different signature-based anti-virus programs. Our combined kernel was the overall winner with an accuracy of 96.41%. The best n-gram method with n=3, L=2500 had an accuracy of 82.15%. It is interesting to note that all machine-learning-based methods easily beat the best signature-based anti-virus, which had an accuracy of 73.32%.

Our novel method extends the n-gram methodology by using 2-grams to condition the transition probabilities of a Markov chain, and then treats that Markov chain as a graph. Taking the Markov chain as a graph allows us to utilize the machinery of graph kernels to construct a similarity matrix between instances in our training set. We use two distinct measures of similarity to construct our kernel matrix: a Gaussian kernel, which measures local similarity between the graphs' edges, and a spectral kernel, which measures global similarity between the graphs. Given our kernel matrix, we can then train a support vector machine to perform classification on new test data.

[1] Symantec White Paper, *Internet Security Threat Report* **16,** (2011).
[2] Christodorescu, M. and Jha, S., *Proc 12th USENIX Secur Symp* (2003).
[3] Reddy, D. and A. Pujari, *J Comput Virol* **2,** 231 (2006).
[4] Dinaburg, A. et al., *Proc 15th ACM Conf Comput Comm Security* (2008).
[5] Quist, D. et al., *J Comput Virol* **7,** 121 (2011).
[6] Chung, F.R.K., *Spectral Graph Theory,* American Mathematical Society (1997).

**Funding Acknowledgments**
DOE NNSA, Cybersecurity Program