SEMANTIC EMBEDDINGS FOR PROGRAM BEHAVIOR PATTERNS

Alexander Chistyakov[1], **Ekaterina Lobacheva**[1,2], **Arseny Kuznetsov**[1], **Alexey Romanenko**[1]
[1]Detection Methods Analysis Group, Kaspersky Lab
[2]Faculty of Computer Science, National Research University Higher School of Economics
Moscow, Russia
{Alexander.Chistyakov, Ekaterina.Lobacheva, Arseny.Kuznetsov, Alexey.Romanenko}@kaspersky.com

ABSTRACT

In this paper, we propose a new feature extraction technique for program execution logs. First, we automatically extract complex patterns from a program's behavior graph. Then, we embed these patterns into a continuous space by training an autoencoder. We evaluate the proposed features on a real-world malicious software detection task. We also find that the embedding space captures interpretable structures in the space of pattern parts.

1 INTRODUCTION

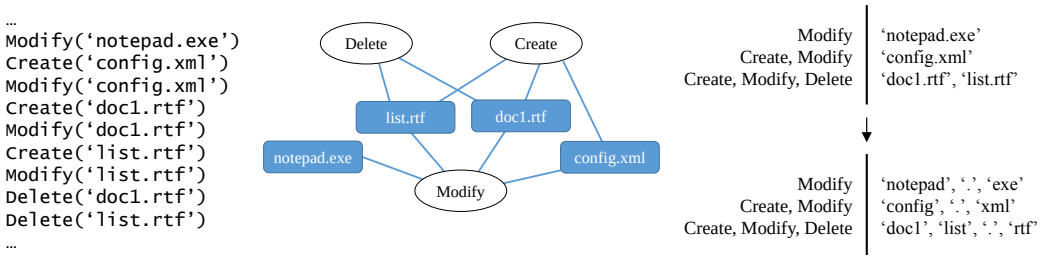
Malware, or malicious software, is a key element of cyberattacks that damages companies and individuals worldwide and benefits criminals. Nowadays, malware is concealed using obfuscation, encryption, anti-emulation and other techniques, making detection of malware significantly harder. Classifying whether a previously unseen file is a malware or a benign program is an important challenge for cybersecurity companies. Employing machine learning methods for this problem is a promising area of research.

The two broad classes of malware analysis techniques are *static* and *dynamic*. The former's methods operate on raw binary files, leading to significant issues with analysis of encrypted and obfuscated files. The latter's approach consists of executing the binary file in a controlled environment and monitoring its behavior. The dynamic approach is more time- and resource-consuming, but it provides higher accuracy. Behavior monitoring results can typically be presented as a log of the observed system events or API calls (function name, arguments, and, optionally, a return value). Currently, the most popular approach for feature extraction from such logs is to construct a set of different indicator features such as n-grams of events or links between APIs and their arguments (Bayer et al. (2009); Berlin et al. (2015); Huang & Stokes (2016); Salehi et al. (2017)). Some other classification methods apply recurrent neural networks to a sequence of notes in a log (Pascanu et al. (2015); Kolosnjaji et al. (2016)). However, the latter approach is sensitive to the mixing of lines in a log caused by multiprocessing, intentional obfuscation by malware, and difference in the execution environments.

In this paper we propose a new feature extraction technique for logs that is based on specific behavior graphs. We consider a graph as a union of behavior patterns (specific subgraphs) and construct a feature representation of a log by combining feature vectors of these patterns. To extract a compact and meaningful continuous feature representation for behavior patterns, we train an autoencoder. In the experiments, we show that the log representation constructed by our technique provides high classification accuracy on a large real-world dataset. In addition, we illustrate the ability of our model to automatically capture interpretable structure in the space of pattern parts similarly to the word2vec model (Mikolov et al. (2013)).

2 FEATURE REPRESENTATION FOR LOGS

In this paper, a log means a sequence of all system events that occurred during program execution alongside with their arguments. A toy example of such a log is presented in Figure 1a. Each line of



(a) Example of a log. (b) Behavior graph. (c) Extracted patterns.

Figure 1: Graph representation of a log and pattern extraction process.

a log corresponds to one system event and contains the type of the event and one or more arguments (file names, URLs, memory addresses, etc.).

Graph representation of a log Because the sequence representation of logs is unstable, we represent them with behavior graphs. Such representation does not rely on the event order but still captures all event-argument interactions. We define a behavior graph as a bipartite graph whose nodes correspond to event types and arguments occurred in the log. Two nodes are connected with an edge if and only if the corresponding event type and argument occur together in the same system event in the log. The graph constructed for the toy example log is presented in Figure 1b.

Lots of papers on representation learning for graphs have appeared in the last few years (Yanardag & Vishwanathan (2015); Grover & Leskovec (2016); Narayanan et al. (2016)); however, they are all limited by a finite node label space. In our problem, we deal with a very diverse space of node labels because of the natural variety of the arguments such as file paths, URLs, and so forth.

To construct a feature representation of a log we extract patterns of program behavior from the graph, build an embedding of these patterns into the space $\mathbb{R}^D$ and then combine separate feature vectors into a graph description.

Behavior pattern extraction The set of the event types that share the same argument in the graph represents some pattern of the program behavior. Furthermore, it is typically important to capture the fact that a sequence of system events is repeated with different arguments in a log. For example, malware could continuously modify items in the same subtree of a file system or try to connect to many different hosts in a row. Arguments themselves also matter: It is one thing when the program downloads and runs a Windows update and completely another when it starts an unknown **.exe* file from a suspicious server. Therefore we define a behavior pattern as the set of system events and arguments such that all of the events share all of the arguments, and each argument corresponds only to these events and nothing else. To extract patterns from the graph, we first find all adjacent event types for each argument and then combine arguments with the same event sets into a single pattern.

At this stage, the selected patterns may be unique, but their arguments may have a lot in common with some other previously observed arguments. For example, the particular file name `C:\Windows\374683.ini` may occur only once in the whole dataset, but the disk name *C*, folder name *Windows* and file extension *ini* are very common. Therefore we propose splitting each argument into a set of tokens by separators (such as '://','.',':', etc.). Here, separators are also considered to be tokens because the type of the argument can be determined from them. Pattern extraction process for the toy example log is illustrated in Figure 1c.

As a result, each pattern can be represented as a sparse binary vector of length $M + K$, where the first M features correspond to event types and the next K features correspond to tokens. Here M is the number of all different event types that exist in the logs, and it is fixed by the logging system. K denotes the number of the most frequent tokens observed in the training dataset, and it can be set manually.

Pattern embeddings To extract a compact and meaningful feature representation for behavior patterns we train an autoencoder model. The encoded representation $a(x)$ and the reconstruction $\hat{v}(x)$ of a pattern x are obtained from its sparse binary representation $v(x)$ as follows:

$$a(x) = Wv(x) + b, \quad \phi(x) = \mathrm{ReLU}(a(x)), \quad \hat{v}(x) = \sigma(V\phi(x) + c), \tag{1}$$

where W and V are trainable weight matrices of size $(M+K) \times D$ and $D \times (M+K)$ respectively, b and c are trainable bias vectors of length D and $(M+K)$ respectively, $\mathrm{ReLU}(y) = \max(0, y)$ and $\sigma(y) = 1/(1+\exp(-y))$. Then, the reconstruction is compared with the original binary feature vector as follows:

$$l(x) = -\frac{1}{|P|}\sum_{i \in P} \log(\hat{v}(x)_i) - \frac{1}{|N|}\sum_{i \in N} \log(1 - \hat{v}(x)_i), \tag{2}$$

where P is the set of non-zero elements in $v(x)$, N is a random subset of zero elements in $v(x)$ and $|\cdot|$ is the cardinality of the set. We take only a small subset of zero elements for efficiency, similar to the negative sampling technique (Mikolov et al. (2013)). Training proceeds by optimizing the sum of reconstruction cross-entropies across the training set using Adam (Kingma & Ba (2015)).

To construct a fixed-size feature representation for a log, we combine the encoded feature vectors $a(x)$ of all patterns from this log by applying the element-wise min, max, and mean functions. As a result, for each log, we have a final feature vector of length $3D$.

3 Results/Experiments

Comparison of different dynamic malware detection methods is difficult because researchers use different sandboxes to collect their data, and none of the datasets are publicly available. To evaluate our log representation, we collected 4 964 506 malicious and 3 145 853 benign logs from our in-lab sandbox and a set of user-managed machines, and randomly split our data into train (70%) and test (30%) sets.

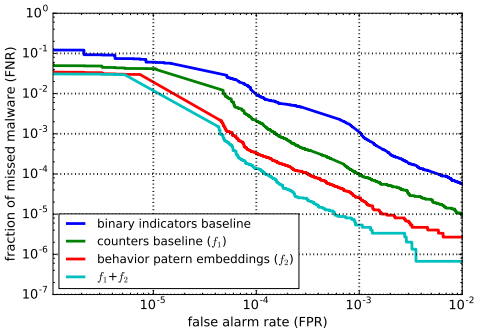


Figure 2: Comparison of feature representations of logs (f_2 is ours).

The majority of existing articles on dynamic malware detection are based on constructing a set of indicator behavior features, so we selected the set of the most frequent groups of events that share the same argument from our training data, and use indicator features constructed from them as a baseline feature representation. We also try to use counters instead of binary indicators. We train the XGBoost model (Chen & Guestrin (2016)) with 300 trees of depth 30 as a classifier to compare our features to baselines. Figure 2 shows that the usage of our semantic features reduces the number of missed malicious samples several times while keeping an extremely low false alarm rate. A combination of our semantic features and baseline counter features provides even better performance.

Table 1: Synonyms locality embedding

TOKEN	TOP-5 NEAREST NEIGHBOURS
word	excel, dotm, outlook, machine, open
com	www, ://, http, net, ru
jpg	png, gif, css, xml, html
js	txt, htm, ie5, css, cookies
34	36, 42, 56, 38, 35
3960	3964, 3972, 3952, 3968, 3956

Table 2: Arithmetic operations on tokens

TOKEN'S EXPRESSION	RESULT
'word' - 'doc' + 'xls'	'excel'
'video' - 'mp4' + 'bmp'	'image'
'programfiles' - 'exe' + 'dll'	'system32'
'https' - 'http' + '80'	'443'
'google' - 'chrome' + 'firefox'	'mozilla'
'(' - ')' + ']'	'['

In order to get some intuition about how our behavior pattern embeddings work, we study the structure of the token embedding space, obtained from rows of the matrix W. Table 1 shows that tokens that are similar in the vector space are also semantically similar: they form groups of program names, file extensions, and decimal constants. Another interesting point of the obtained embedding is that it keeps semantic relations between pairs of tokens in the manner of the canonical word2vec model (Mikolov et al. (2013)). Table 2 presents some of these relations such as the binding of a file format to its typical folder or editor program name, TCP/IP port number to the corresponding web-protocol name, and programming product name to its vendor.

REFERENCES

Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pp. 8–11. Citeseer, 2009.

Konstantin Berlin, David Slater, and Joshua Saxe. Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pp. 35–44. ACM, 2015.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2016.

Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 855–864. ACM, 2016.

Wenyi Huang and Jack W. Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 399–418. Springer, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the the 3rd International Conference for Learning Representations*, 2015.

Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, pp. 137–149. Springer, 2016.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. Curran Associates, Inc., 2013.

Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *CoRR*, abs/1606.08928, 2016.

Razvan Pascanu, Jack W. Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. Malware classification with recurrent networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1916–1920. IEEE, 2015.

Zahra Salehi, Ashkan Sami, and Mahboobe Ghiasi. MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values. *Engineering Applications of Artificial Intelligence*, 59:93–102, 2017.

Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374. ACM, 2015.