Learning System Call based Language Model for Malicious Process Detection

Naman Patel[1], Manuel Serrano Rebuelta [1]

Abstract—**Malicious software in the form of computer viruses, Trojan horses, bots, and Internet worms like adware, spyware, and ransomware poses a serious threat to computer security. The amount of different malwares and its possible variants are numerous which makes classical condition based or signature based approaches ineffective. Although these malicious software are plentiful, these variants of malware families share typical behavioral patterns reflecting its origin and purpose. In this paper we study the capability of various language modeling based approaches to extract these behavioral pattern for system call based malware detection. A detailed analysis of the effectiveness of various language modeling based features, namely, Bag of Words, Term Frequency - Inverse Document Frequency and word representation is presented along with their performance using classifiers based on Naïve Bayes, SVM and logistic regression on the MALREC dataset.**

I. Introduction

As society becomes more and more dependent on computing systems, detection of malicious software becomes important as these can cause strong financial damage. Current approaches like anti-virus softwares provide some protection against malware, but are growing increasingly ineffective for the problem. Current anti-virus technologies use a signature-based approach, where a signature is a set of manually crafted rules in an attempt to identify a small family of malware. These rules are generally specific, and cannot usually recognize new malware even if it uses the same functionality. This approach is insufficient as most environments will have unique processes that will have never been seen before and millions of new malware samples are found every day. The limitations of signatures have been recognized by the anti-virus providers and industry experts for many years. The need to develop techniques that generalize to new malware would make the task of malware detection a seemingly perfect fit for machine learning, though there exist significant challenges.

To build a malware detection system, we must first determine a feature set to use. One intuitive choice is to use features obtained by monitoring program execution (APIs called, instructions executed, IP addresses accessed, etc). To this end recently, a sandbox system, MALREC was introduced. The Malrec sandbox is built on top of PANDA [1], a whole-system dynamic analysis platform. The key feature of PANDA for our purposes is deterministic record and replay. Record and replay, as implemented in PANDA, captures compact, whole-system execution traces by saving a snapshot of the system state at the beginning of the recording and then recording all sources of non-determinism: interrupts, input from peripherals, etc. At replay time, the snapshot is loaded and the system is run with all peripherals disabled; the stored non-deterministic events are replayed from the log file at the appropriate times. This ensures that as long as we have accounted for all sources of non-determinism, the replayed execution will follow the exact path as the original execution. Specifically, for this paper we extract system call information from this dataset with malware and benign activity label and perform behavioral analysis on the dataset based on language modeling.

Behavioral analysis techniques use characteristics of executing software to identify potential malware. One such technique is system call analysis, wherein malicious behaviors are identified by their system call traces [2]–[4]. System calls are requests for operating system services, such as memory and filesystem access. This study describes a malware detection system that uses system call analysis to detect malware that evade traditional defenses and learn malware signatures automatically [5], [6]. The system is intended to supplement existing defenses such as antivirus software by identifying compromised hosts. The specific configuration of the system is determined through experimental evaluation of multiple feature extraction and detection strategies.

The main contribution of this study is the evaluation and comparison of detection and feature extraction techniques. Behavioral analysis based on language modeling features based on Bag of Words, TF-IDF and word vector representation based on system call names and system call arguments are studied and detection techniques based on Naïve Bayes, SVM and logistic regression are analyzed in detail.

This paper is organized as follows. The dataset, and feature extraction techniques are discussed in Section II. Various detection algorithms for malicious process detection are discussed in Section III. Section IV describes our experimental results on the system call MALREC dataset with malware activity ranges. Section V concludes the paper and gives hints about future directions of our work.

II. Problem Formulation

A. MALREC

MALREC [7] is a malware sandbox system that contains 66,301 malware recordings collected over a two-year period, between December 7, 2014 and December 3, 2016. The samples are provided in a daily feed by the Georgia Tech Information Security Center (GTISC) and come from ISPs and antivirus vendors.

[1]All the authors are with the Department of Electrical and Computer Engineering, NYU Tandon School of Engineering, 5 MetroTech Center, Brooklyn, NY, USA. {nkp269, msr542}@nyu.edu.

MALREC uses PANDA's whole-system deterministic record and replay to capture high-fidelity, whole-system traces of malware executions with low time and space overheads. Record and replay, as implemented in PANDA, captures compact, whole-system execution traces by saving a snapshot of the system state at the beginning of the recording and then recording all sources of non-determinism, such as interrupts, input from peripherals, etc.

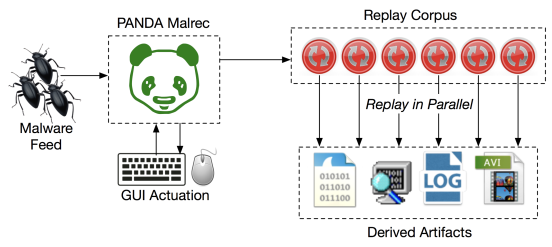


Fig. 1. The MALREC recording system. Malware recordings are fed into PANDA sandbox along with the GUI actuation. The replay can then be used to produce different kinds of derived datasets for behavioral and retrospective analysis for malicious process detection based on network logs, memory snapshots, and system calls.

1) Dataset: Deterministic replays are generated using PANDA based MALREC sandbox. These replays are utilized to generate various datasets for behavioral analysis of malicious processes. The malicious processes have ground truth label through various anti-virus softwares using honeypots. These datasets include:

- Textual data in memory accesses.
- Memory snapshots.
- Network activity in PCAP form.
- Virus classification among 65 families.
- System calls.

2) System Call - Activity Ranges dataset: In our study, we use two different sources of data comprised in the dataset. These are:

- System call traces that contain instruction number, address space identifier, name and arguments:

```
32525822 3f9b8360 NtUserGetProp 00010082
32529674 3f9b8360 NtDelayExecution 00000000
32539932 3f9b8360 NtUserCallNoParam 00000003
32540622 3f9b8360 NtUserPeekMessage 032cfe84
```

- Activity ranges of instruction number in which the malware was being executed, associated with its corresponding file of system calls:

```
3136382089        3141569433
3156872546        3156878180
3156889788        3156898702
3209194758        3209770663
```

In other words, there are two sources of data for each malware recording: the sequence of all system calls, and the ranges in these system calls in which the malware was being executed.

We make use of both parallelly, and for each of these data sources, we use a subset of 200 train files and 100 test files drawn from the 66,301 MALREC recordings, with a total of 498 different function names, and up to 8,336,504 different system calls when function arguments are concatenated to the names.

B. Feature extraction from system call traces

In order to create a formatted dataset to perform feature extraction, we created Algorithm 1, that labels data from the two files we previously mentioned.

Algorithm 1 Malware labeling process

```
Require: $systemcalls$, folder with all system call files
 1: Initilize empty file $Data$
 2: $previousline \leftarrow 0$
 3: for each $SC \in systemcalls$ do
 4:     $AR \leftarrow$ Activity ranges file that corresponds to $SC$
 5:     for each $line$ in $AR$ do
 6:         $Benign \leftarrow$ system call names within $line$ in $SC$
 7:         $Malware \leftarrow$ system call names between $line$ and $previousline$ in $SC$
 8:         $Data.insert(Benign, 0)$
 9:         $Data.insert(Malware, 1)$
10:         $previousline \leftarrow line$
11:     end for
12: end for
13: Save $Data$ as .csv file for further use
```

1) Bag of Words (BoW) Features: For malware trace classification, a Bag of Words model is a dictionary in which the keys are all different names of system calls found in the given set of traces, and values are the number of occurences for each key. Similarly, a Bag of n-grams model is the same representation where names of system calls are replaced by n-grams, which are system calls occurring contiguously in a trace, extracted using a sliding window. In our study, we will be using BoW as well as Bag of 2-grams.

2) TF-IDF Features: TF-IDF is a feature scaling technique that creates a vectorized version of each trace from the dot product of the term frequency (TF), computed directly from x, and the inverse document frequency (IDF), computed from the all set of documents $\mathcal{X}$. More formally:

$$TF - IDF(x, \mathcal{X}) = TF(x) \odot IDF(\mathcal{X}) \tag{1}$$

where $TF(x)$ can be raw or logarithmic term frequency of the document, such that:

$$TF(x) = x \text{ or } TF(x) = \log(x+1) \tag{2}$$

and $IDF(\mathcal{X})$ is defined as:

$$IDF(\mathcal{X}) = \log \frac{1+n}{1+d} + 1 \tag{3}$$

where n is the number of traces in $\mathcal{X}$ and d is the vector counting the traces in which each n-gram appears.

3) Word vector representation: Various word representation based techniques have been proposed for learning features from documents [8]–[12]. We use the supervised learning based technique [10], [11] with fastText for extracting feature representation based on two different inputs. These are:

- Features based on system calls names only: The system call function names are extracted from a given trace and added to the vocabulary. For each word in the vocabulary a vector of a fixed size (100 in our case) is generated randomly from an uniform distribution. The size of the vocabulary for the given training set is 498. Additionally, in order to add subword information character n-gram vectors are also generated and have their own vector representation as explained in the next section.
- Features based on system calls names and arguments: The system call function names and their corresponding arguments are concatenated and added to the vocabulary. Similar to the system call name based features, for each word in the vocabulary a randomly generated vector from an uniform distribution. The size of the vocabulary in this case escalates to 8,336,504 for the training set.

III. Proposed Framework

In this section we will explain and elaborate on all models and approaches we have used and compared. All of the following approaches have been used with the same train and test sets, in order to accurately compare performances.

As previously mentioned, in this study we have used several machine learning basic classification methods for which we have used both BoW and TF-IDF data representations. Also, we have used fastText based classification. These methods are:

A. Naïve Bayes classifiers

Naïve Bayes classifiers are a family of machine learning classification algorithms based on the Bayes Theorem that assumes strong independence assumptions between the features (hence "naïve").

$$P(Malware|x) = \frac{P(x|Malware)P(Malware)}{P(x)} \quad (4)$$

In our experiments, each malware or benign trace is represented as a vector $x = \langle x_1, ..., x_m \rangle$, where $x_1, ..., x_m$ is the vectorized version of sequence of system calls, given by the BoW or TFIDF feature extractor [13]. Within this family of classifiers, we will be focusing on the following two classifiers:

1) Bernoulli Naïve Bayes: A probabilistic classifier based on the Bayes Theorem that assumes independence among the features, and also assumes that all features are binary [14]. Thus, we have

$$P(x|Malware) = \prod_{i=1}^{M} p_{i,Malware}^{x_i}(1 - p_{i,Malware})^{1-x_i} \quad (5)$$

where $p_{i,Malware}$ represents the probability of class malware generating the term x_i.

2) Multinomial Naïve Bayes: A probabilistic classifier based on the Naïve Bayes assumption that it assumes that the probability of a term being generated by a malware class follows a multinomial distribution. More formally, this assumption leads to:

$$P(x|Malware) = \frac{(\sum_{i=1}^{M} x_i)!}{(\prod_{i=1}^{M} x_i)!} \prod_{i=1}^{M} p_{i,Malware}^{x_i} \quad (6)$$

B. Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning framework that aims to classify two different input non-linearly separable vectors [15] by mapping them into a higher dimensional feature space, and constructing a linear decision hyperplane for classification. In our study, we make use of a Linear SVM that seeks a hyperplane $w^T x$ to optimally separate two sets of data points, which are vectorized versions of malware and benign traces. Parameters w are calculated using SGD from data with labels $y_i \in \{-1, 1\}$. The objective function minimized is the following:

$$\max_{w}\{0, y_i(w^T x_i + b)\} + \alpha||w||_2 \quad (7)$$

where the first term is a Hinge Loss that aims to minimize mis-classifications, and the second term acts as an L_2 regularizer trying to maximize the classification margin.

C. Logistic Regression (LR)

Logistic Regression is a statistical model widely used for binary classification tasks with a strong probabilistic interpretation. The aim for using LR in our study is to determine the probability of a trace being malware, given the sequence of system calls. The objective function for LR uses the following logarithmic loss:

$$L(y_i, \hat{y}_i) = \log(1 + \exp(y_i \hat{y}_i)) \quad (8)$$

where y_i represents the ground truth label, and $\hat{y}_i$ the corresponding prediction, that is given by the following probability:

$$P(Malware|x) = \frac{1}{1 + e^{-w^T x}} \quad (9)$$

D. FastText based Classification

Figure 2 shows a simple linear model with rank constraint. The first weight matrix A is a look-up table over the words. The word representations are then averaged into a text representation, which is in turn fed to a linear classifier. The text representation is an hidden variable which can be potentially be reused. This architecture is similar to the continuous bag of words model, where the middle word is replaced by a label. Each word in the vocabulary is randomly sampled from a normal distribution and is of length 100. The number of words to be used as input for the label is called the context window which after random search is selected to be 5. Thus, vector representation of the 10 words (system calls) are multiplied by matrix A and averaged to get the hidden vector representation. In order to learn, the representation for the sequence of labels, the sequence is converted to multiple context window and we maximize the probability of the label given all the context which is given by the following equation:

$$\sum_{t=1}^{T}[\log p(l_t|C_t)] \tag{10}$$

where C_t is the t^{th} context sequence from the whole sequence for label l_t.

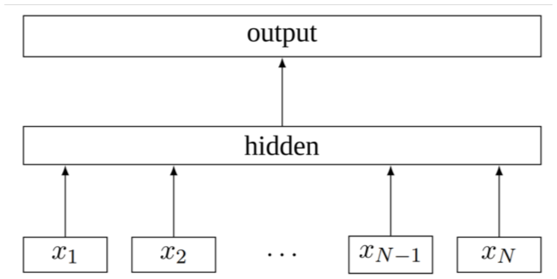


Fig. 2. Architecture for learning word representation with system call sequence with N n-gram features $x_1, \ldots, x_N$ as inputs and malware label as the output. The features from each input n-grams are embedded and averaged to form the hidden variable which is in turn fed to a linear classifier.

1) Negative Sampling: In order to learn efficiently with huge number of labels, we learn through negative sampling where the correct word is learned in contrast with a set of sampled negative candidates. The original loss function which maximizes the probability of the label given the system call sequence with N contexts is replaced with the following:

$$\log(1+e^{-s(l,C)}) + \sum_{n\in N_C}\log(1+e^{s(n,C)}) \tag{11}$$

where N_C is a set of negative examples sampled from the vocabulary. The objective function maximized by the fastText model is obtained by replacing the log probability in equation 10 by the quantity defined in Equation 11, i.e.:

$$\sum_{t=1}^{T}[\log(1+e^{-s(l_t,C_t)}) + \sum_{n\in N_{C_t}}\log(1+e^{s(n,C_t)})] \tag{12}$$

2) Word Subsampling: Similar to standard text corpuses, most of system calls of the MALREC dataset belong to a small subset of the entire vocabulary of system calls. Considering all the occurences of system calls equally would lead to overfit the parameters of the model on the representation of the most frequent system calls, while underfitting on the rest. In order to counter this imbalance between the rare and frequent system calls, a simple subsampling approach is used where each word w_i in the training set is discarded with probability computed by the formula

$$P(w_i) = 1 - \sqrt{\frac{t}{f_{w_i}}} \tag{13}$$

where f_{w_i} is the frequency of word wi, and t>0 is a is a chosen threshold (chosen as 10^{-4} after search over 10^{-1} to 10^{-6}). This subsampling formula aggressively subsamples system calls whose frequency is greater than t while preserving the ranking of the frequencies which accelerates learning and significantly improves the accuracy of the learned vectors of the rare words.

3) Position-dependent Weighting: In the vanilla implementation of the representation vector based classifier, the representations for each system call in a sequence are averaged to learn the hidden representation which makes it oblivious to the position of each word. Thus, to implicitly learn representation for both the system call and its position in the sequence, we learn the position representation and use them to reweigh the word vectors. This position dependent weighting offers a richer context representation at a minimal computational cost. Each position p in a context window is associated with a vector d_p. The context vector v_C reweighted by the position vectors of the word w_t is given by

$$v_C = \sum_{p\in P} d_p \odot u_{t+p} \tag{14}$$

where P are the set of context window relative positions $[-c, \ldots, -1, 1, \ldots, c]$ and u_t is the position vector for position t.

4) Phrase Representations: In addition to position dependent weighing, we take the system call order into account by word n-gram representation [9]. The n-grams representations are selected by iteratively applying a mutual information criterion to bigrams for consequtive system calls in the sequence. Then, in a data pre-processing step we merge the system calls in a selected n-gram into a single token. This pre-processing step is repeated several times to form longer n-gram tokens upto a certain max lenght l. Unigrams with high mutual information are merged only with a probability of 50%, thus a significant number of unigram occurrences are kept. After random search of max length l of n-gram sequence from 2 to 6, we selected an n-gram sequence of max legth 3. Thus, a maximum of three unigrams can be merged to form a single token. A fast and memory efficient mapping of the n-grams representation is implemented by using the hashing trick [16] with 2 million bins. Thus, these phrase representation effectively improve the quality of the word vectors.

5) Sub-word Information: In order to incorporate internal structure of system call names, sub-word information which is breaking the name into sub-names is used. This information could be useful for computing representation of similar system call names to find their internal representation [17]. A simple yet effective approach is to enrich the word vectors with a bag of character n-gram vectors that is either derived from the singular value decomposition of the co-occurence matrix or directly learned from a large corpus of data. We use the latter approach where each word is decomposed into its character n-grams N and each n-gram n is represented by a vector x_n. The word vector is then simply the sum of both representations,

$$v_w + \frac{1}{\|N\|}\sum_{n\in N} x_n \tag{15}$$

We efficiently store and compute these vectors using the hashing trick. A random search was performed to select the minimum and maximum lenght of character n-gram and was found to be 12 characters and 32 characters respectively.

IV. Experimental Studies

Further we report results for all models trained on a subset of 200 train files and 100 test files drawn from the 66,301 MALREC recordings, with a total of 498 different function names, and 8,336,504 different system calls with function arguments concatenated. The metrics we use are:

$$Accuracy = \frac{TP + TN}{P + N} \tag{16}$$

$$Precision = \frac{TP}{TP + FP} \tag{17}$$

$$Recall = \frac{TP}{TP + FN} \tag{18}$$

$$F1score = \frac{Precision \cdot Recall}{Precision + Recall} \tag{19}$$

For both Naïve Bayes based classifiers, SVM and Logistic Regression, BoW and TF-IDF with 1-grams and 2-grams have been used. SVM has been used along with a linear kernel, and both SVM and LR have been cross-validated until finding the optimal regularization term in $C = 10000$.

As the table below shows, fastText feature extraction with system call names only achieves a maximum F1 score of 93.9%. The fact that fastText feature extraction achieved the best performance was expected because for each word, it creates an embedding with much richer information than BoW or TF-IDF. However, it is worth noting that using the arguments of system calls produces a lower F1 score (90.6%). This might happen because arguments are particular memory locations that add no extra valuable information and follow no pattern.

V. Conclusions and future work

Various learning based language modeling approaches were analyzed for the task of malicious process detection based on system call. Linear classification model based on continuous bag of words features with larger context performed the best.

Thus, the above vector representation can be further improved by recurrent architectures like BiLSTM which induce long term context and QRNN [18] which exploit both context and parallelism.

Additionally, we would also like to generate adversarial examples for these malware detection classifiers which are the sequence of benign system calls to be added to a malware process which makes the classifier detect the sequence as benign.

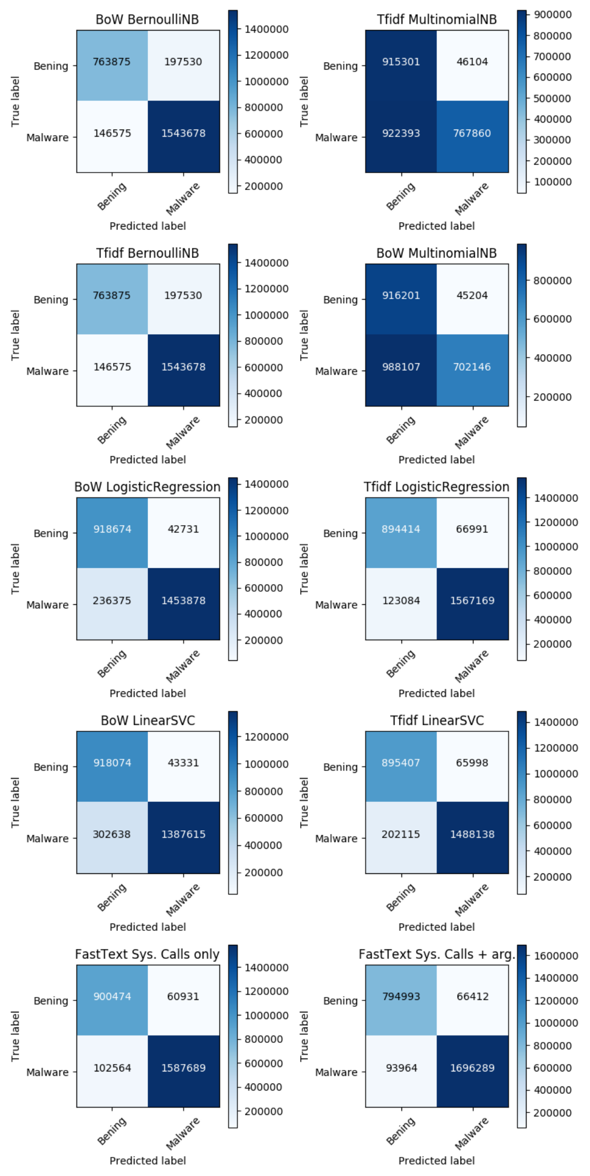


Fig. 3. Confusion matrices for different classifiers based on various feature representations.

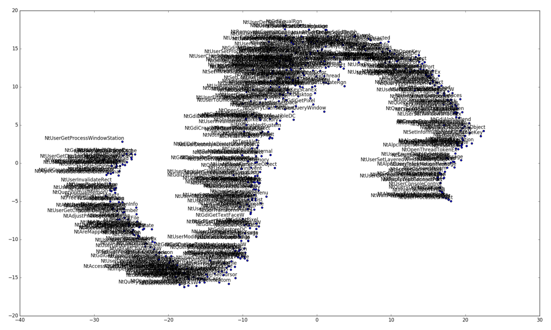


Fig. 4. t-distributed stochastic neighbor embedding (t-SNE) visualization of vector embeddings generated from the learned system call name representations.

Tokenizer	Classifier	Accuracy	Precision	Recall	F1 score
BoW	Bernoulli NB	0.87	0.913	0.887	0.869
	Multinomial NB	0.61	0.415	0.94	0.599
	SVM	0.87	0.821	0.97	0.872
	Logistic Regression	0.895	0.86	**0.971**	0.896
TF-IDF	Bernoulli NB	0.87	0.913	0.887	0.869
	Multinomial NB	0.635	0.454	0.943	0.628
	SVM	0.899	0.88	0.958	0.9
	Logistic Regression	0.928	0.927	0.959	0.929
FastText	System calls only	**0.938**	0.939	0.963	**0.939**
	System calls + arguments	0.907	**0.949**	0.908	0.906

TABLE I
PERFORMANCE COMPARISON OF VARIOUS MODELS ON THE MALREC SYSTEM CALL BASED MALICIOUS PROCESS DETECTION DATASET.MORE DETAILS

REFERENCES

[1] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan, "Repeatable reverse engineering with PANDA," in *Proceedings of the 5th Program Protection and Reverse Engineering Workshop, PPREW@ACSAC, Los Angeles, CA, USA, December 8, 2015*, 2015, pp. 4:1–4:11.

[2] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015, Vancouver, BC, Canada, August 3-5, 2015*, 2015, pp. 119–124.

[3] K. Dam and T. Touili, "Automatic extraction of malicious behaviors," in *11th International Conference on Malicious and Unwanted Software, MALWARE 2016, Fajardo, PR, USA, October 18-21, 2016*, 2016, pp. 47–56.

[4] R. Canzanese, S. Mancoridis, and M. Kam, "Run-time classification of malicious processes using system call analysis," in *10th International Conference on Malicious and Unwanted Software, MALWARE 2015, Fajardo, PR, USA, October 20-22, 2015*, 2015, pp. 21–28.

[5] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1–6:42, 2012.

[6] R. S. Pirscoveanu, S. S. Hansen, T. M. T. Larsen, M. Stevanovic, J. M. Pedersen, and A. Czech, "Analysis of malware behavior: Type classification using machine learning," in *2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), London, United Kingdom, June 8-9, 2015*, 2015, pp. 1–7.

[7] G. Severi, T. Leek, and B. Dolan-Gavitt, "Malrec: Compact full-trace malware recording for retrospective deep analysis," 2018, submitted for review.

[8] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, pp. 1532–1543.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *In proceedings of the Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, 2013, pp. 3111–3119.

[10] E. Grave, T. Mikolov, A. Joulin, and P. Bojanowski, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, 2017, pp. 427–431.

[11] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, vol. 2, 2017, pp. 427–431.

[12] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, 2008, pp. 160–167.

[13] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial naive bayes for text categorization revisited," in *AI 2004: Advances in Artificial Intelligence, 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, December 4-6, 2004, Proceedings*, 2004, pp. 488–499.

[14] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes - which naive bayes?" in *CEAS 2006 - The Third Conference on Email and Anti-Spam, July 27-28, 2006, Mountain View, California, USA*, 2006.

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[16] K. Q. Weinberger, A. Dasgupta, J. Langford, A. J. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, 2009, pp. 1113–1120.

[17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *TACL*, vol. 5, pp. 135–146, 2017.

[18] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," *arXiv preprint arXiv:1611.01576*, 2016.

NYU POLYTECHNIC SCHOOL OF ENGINEERING

Learning System Call based Language Model for Malicious Process Detection

Naman Patel and Manuel Serrano

ABSTRACT

An analysis of language model learning based methods for malicious process detection using system call information.

The models are trained on the MALREC [1] dataset which provides a labeled dataset of malicious and benign processes.

The dataset comprises of a sequence of system calls and their corresponding arguments.

In our study we compare the following models on the above dataset:

- Bayesian methods, SVM and Logistic Regression on Bag of Words and Term Frequency - Inverse Document Frequency features
- Linear supervised classification model with continuous bag of words features with various hyperparameters [2].

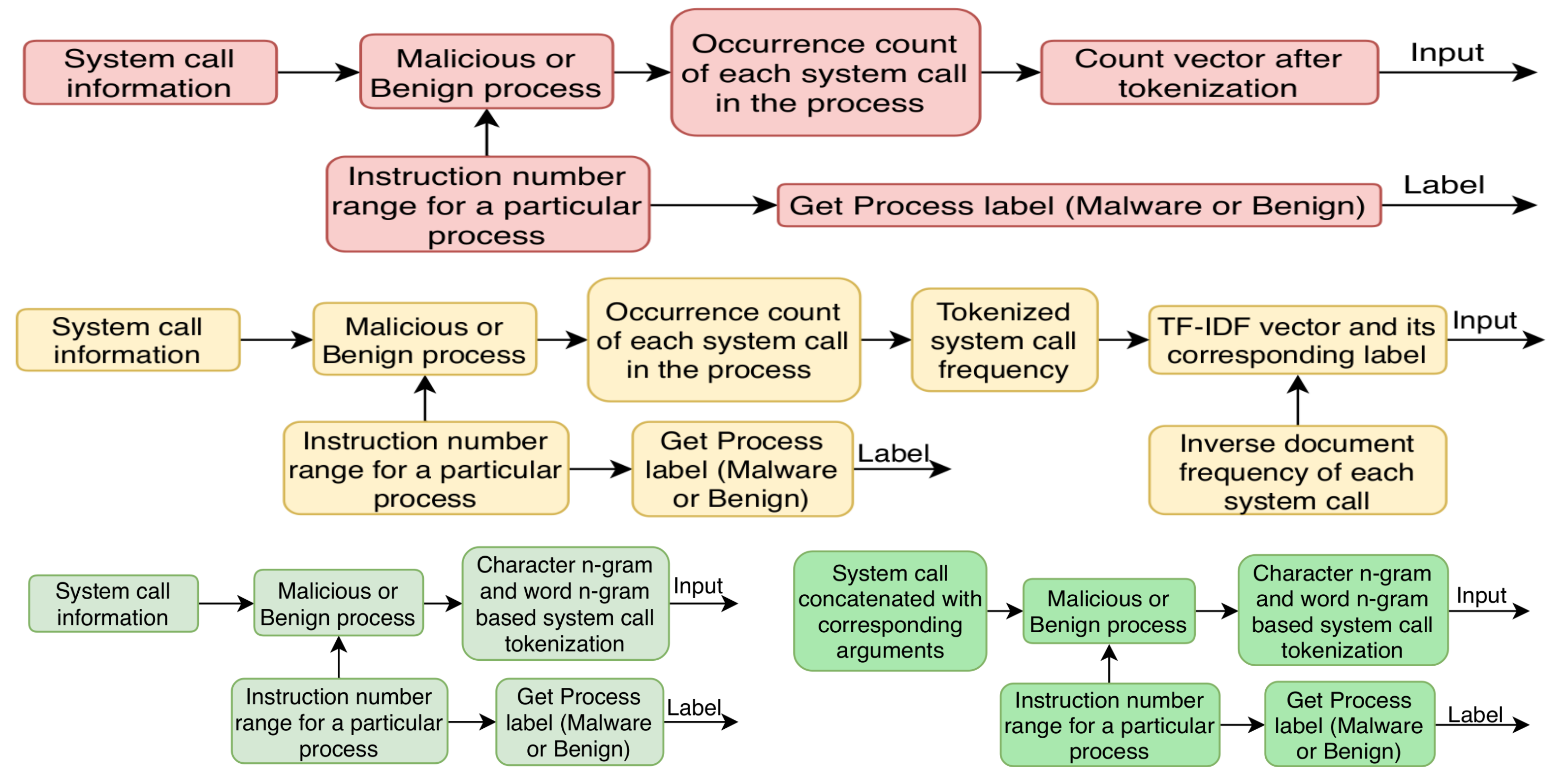
DATASET

Malrec is a sandbox system that contains 66.301 malware recordings, able to deterministically replay system calls, network activity and memory snapshot.

We use a subset of 200 train files and 100 test files, for which we create the following two sets of data:

1. Input: Sequence of system calls name. Output: Malware label
   - 498 words, 1995037 train and 2651658 test instances
2. Input: Sequence of system call name and corresponding arguments concatenated. Output: Malware label
   - 8336504 words, 1995037 train and 2651658 test instances

DATA PREPROCESSING


System call information
Malicious or Benign process
Occurrence count of each system call in the process
Count vector after tokenization
Input
Instruction number range for a particular process
Get Process label (Malware or Benign)
Label
System call information
Malicious or Benign process
Occurrence count of each system call in the process
Tokenized system call frequency
TF-IDF vector and its corresponding label
Input
Instruction number range for a particular process
Get Process label (Malware or Benign)
Label
Inverse document frequency of each system call
System call information
Malicious or Benign process
Character n-gram and word n-gram based system call tokenization
Input
Instruction number range for a particular process
Get Process label (Malware or Benign)
Label
System call concatenated with corresponding arguments
Malicious or Benign process
Character n-gram and word n-gram based system call tokenization
Input
Instruction number range for a particular process
Get Process label (Malware or Benign)
Label

APPROACHES

MACHINE LEARNING BASIC CLASSIFICATION MODELS

We compare four different classification algorithms in which we perform both Bag of Words and Term Frequency. These are:

- Bernoulli Naïve Bayes: $P(Mal|x) = \frac{P(x|Mal) * P(Mal)}{P(x)}$ $\quad P(x|Mal) = \prod_{i=1}^{M} p_{i,s}^{x_i}(1-p_{i,s})^{1-x_i}$
- Multinomial Naïve Bayes: $P(\bar{x}|Mal) = p(D)D!\prod_{i=0}^{M}\frac{(p_{i,s})}{\bar{x}_1!}$
- Support Vector Machine: $\varepsilon = \max(0, 1 - y_i(wx_i + b) + \alpha||w||_2)$
- Logistic Regression: $p_{Mal} = \frac{1}{1 + e^{-(\beta_1 x + \beta_0)}}$

```
33994077 3f9b8120 NtClose 0000055c
33996246 3f9b8120 NtOpenKeyEx 00d5e784 00020019
34004901 3f9b8120 NtDeleteValueKey 00000548 00d5ef6c
34009899 3f9b8120 NtClose 00000548
34011424 3f9b8120 NtClose 000003c8
```

Example of system calls

FASTTEXT

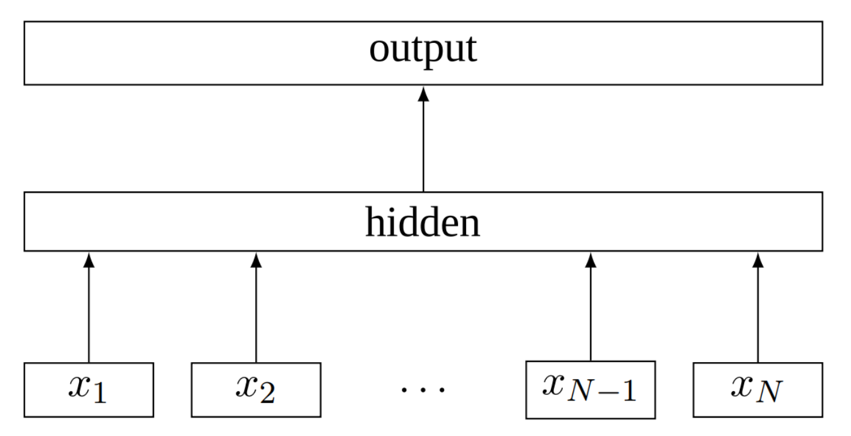


Figure: Model architecture of fastText [3] for a sentence with N n-gram features x1,..., xN. The features are embedded and averaged to form the hidden variable.

The language model is learnt by minimizing the following function:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n))$$

Where *f* is a softmax function, *A* and *B* are the weight matrices of the text representation for the bag of n-gram features x and its corresponding label y.

The word vectors and their corresponding character n-grams word vectors are initialized from a random uniform distribution.

The input bag of n-gram feature for each word is a sum of the word vector and the average of the word's char n-gram vector.

The fastText model was tuned on the following hyperparameters:

- Size of the word vector
- Maximum length of word n-grams
- Minimum and maximum length of character n-grams
- Number of samples used for negative sampling
- Size of the context window

The best model for both the datasets had a vector size, max length of word n-grams, minimum length of character n-grams, number of negative samples and size of the context window of 100, 3, 12, 2, 5 respectively.

RESULTS

		Accuracy	Precision	Recall	F1 score
Bag of words	BernoulliNB	0.87	0.913	0.887	0.869
	MultinomialNB	0.61	0.415	0.94	0.599
	LinearSVC	0.87	0.821	0.97	0.872
	LogisticRegression	0.895	0.86	**0.971**	0.896
TF-IDF	BernoulliNB	0.87	0.913	0.887	0.869
	MultinomialNB	0.635	0.454	0.943	0.628
	LinearSVC	0.899	0.88	0.958	0.9
	LogisticRegression	0.928	0.927	0.959	0.929
FastText	System Calls only	**0.938**	0.939	0.963	**0.939**
	System Calls with arguments	0.907	**0.949**	0.908	0.906

Table: Performance comparison of various models on the MALREC system call based malicious process detection dataset.

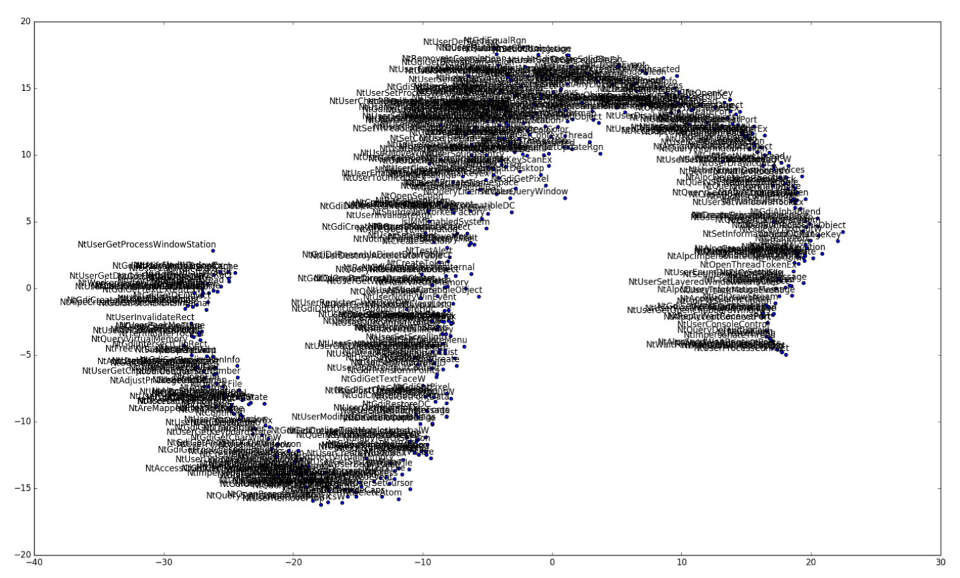


Figure: t-distributed stochastic neighbor embedding (t-SNE) visualization of vector embeddings generated from the learned system call name representations

CONCLUSIONS

Various learning based language modeling approaches were analyzed for the task of malicious process detection based on system call, in which linear classification model based on continuous bag of words features with larger context performed the best. The above vector representation can be used by recurrent architectures like BiLSTM and QRNN [4].

REFERENCES

[1] Severi, G., Leek, T. and Dolan-Gavitt, B., *MALREC: Compact full-trace malware recording for retrospective deep analysis*, Submitted for review, 2018.

[2] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J, *Distributed representations of words and phrases and their compositionality*, In *Advances in neural information processing systems*, pp. 3111-3119, 2013.

[3] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T., *Bag of Tricks for Efficient Text Classification*, In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, vol. 2, pp. 427-431, 2017.

[4] Bradbury, J., Merity, S., Xiong, C., and Socher, R., Quasi-Recurrent Neural Networks, *In Proceedings of the International Conference on Learning Representations, 2017.*